RAPID APPLICATION MOBILIZATION AND DELIVERY FOR SMARTPHONES

A Dissertation
Presented to
The Academic Faculty

by

Cheng-Lin Tsao

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
August 2012

RAPID APPLICATION MOBILIZATION AND DELIVERY FOR SMARTPHONES

Approved by:

Professor Raghupathy Sivakumar,
Advisor
School of Electrical and Computer
Engineering
Georgia Institute of Technology

Professor Douglas M. Blough
School of Electrical and Computer
Engineering
Georgia Institute of Technology

Professor Nikil S. Jayant
School of Electrical and Computer
Engineering
Georgia Institute of Technology

Professor Umakishore Ramachandran
College of Computing
Georgia Institute of Technology

Professor Ghassan AlRegib
School of Electrical and Computer
Engineering
Georgia Institute of Technology

Date Approved: June 2012

DEDICATION

To

my parents Min-Chia Tsao and Chin-Chu Tu,

my wife Hsun-Han Yu

and

my daughter Sophia Y. Tsao.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude to my advisor, Prof. Raghupathy Sivakumar, for his unflagging guidance and support. This dissertation would not have been possible without all the insightful discussion with him. He has given me great freedom in making my own decisions and coming up with research ideas. During my dissertation study, he has acted as an excellent role model of a researcher with intense enthusiasm and drive. His high standards for clear thinking and effective communication are inspiring to me will continue to guide me in my future endeavors.

I would like to thank Profs. Nikil S. Jayant, Douglas M. Blough, Umakishore Ramachandran, and Ghassan AlRegib for serving in my proposal and dissertation committee. I am grateful for their valuable advices and opinions, which helped me improve the quality of this dissertation.

My gratitude extends to the present and past members of the GNAN research group. I appreciate the significant help from Sandeep Kakumanu in multiple research projects. I thank Tae-Young Chang, Zhenyun Zhuang, Sriram Lakshmanan, Yeonsik Jeong, Aravind Velayutham, Shruti Sanadhya, Chao-Fang Shih, Bhuvana Krishnaswamy, Jiechao Wang, and Nishith Agarwal for their valuable feedback and assistance in my dissertation.

Last but not the least, I would like to thank my family: my parents, my wife, and my daughter. Their encouragement, support, and sacrifices have helped me go through the ups and downs of this journey. To them, I dedicate this dissertation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Smartphones form an emerging mobile computing platform that has hybrid characteristics borrowed from PC and feature phone environments. While maintaining great mobility and portability as feature phones, smartphones offers advanced computation capabilities and network connectivity. Although the smartphone platform can support PC-grade applications, the platform exhibits fundamentally different characteristics from the PC platform. Two important problems arise in the smartphone platform: how to mobilize applications and how to deliver them effectively. Traditional application mobilization involves significant cost in development and typically provides limited functionality of the PC version. Since the mobile applications rely on the embedded wireless interfaces of smartphones for network access, the application performance is impacted by the inferior characteristics of the wireless networks. Our first contribution is *super-aggregation*, a rapid application delivery protocol that in tandem uses the multiple interfaces intelligently to achieve a performance that is "better than the sum of throughputs" achievable through each of the interfaces individually. The second contribution is *MORPH*, a remote computing protocol for heterogeneous devices that transforms the application views on the PC platform into smartphone-friendly views. *MORPH* virtualizes application views independent of the UI framework used into an abstract representation called virtual view. It allows transformation services to be easily programmed to realize a smartphone friendly view by manipulating the virtual view. The third contribution is the system design of *super-aggregation* and *MORPH* that achieve rapid application delivery and mobilization. Both solutions require only software modifications that can be easily deployed to smartphones.

CHAPTER I

INTRODUCTION

The rapid growth of smartphones in recent years has embodied the emergence of a new computing platform. Smartphones, which are mobile phones with advanced computation capabilities and network connectivity, possess hybrid characteristics borrowed from PCs and feature phones. On one hand, smartphones provide the same level of portability and mobility as feature phones with the small form factor and battery power of their hardware design. On the other hand, smartphones are significantly improved from feature phones in the following aspects which make the platform more similar to the PC platform. (1) Smartphones run a complete operating system and a full-fledged application platform. The platform enables mobile applications, which are usually called mobile apps, to be easily installed and executed by the user. (2) Smartphones provide a standard TCP/IP protocol stack to allow mobile apps to communicate with application servers in the same way as PCs do, instead of through a more limited protocol such as WAP. (3) Smartphones provide higher computation capabilities in terms of CPU, memory, and storage. (4) Smartphones provide mobile broadband connectivity, and they are typically equipped with multiple wireless interfaces such as Wi-Fi, cellular networks, and Bluetooth. (5) Smartphones provide better input interfaces such as a touch screen, a trace ball, and/or a qwerty keyboard. With those attributes, smartphones have formed a new mobile computing platform that is capable of supporting PC-grade applications, which are applications traditionally implemented in PCs.

While the smartphone platform is capable of supporting PC-grade applications, it has several fundamental characteristics that distinguish it from the PC platform.

(1) As a mobile device, a smartphone has to rely on wireless networks to provide network connectivity. To cater to the requirement of mobile users of ubiquitous and best possible connectivity, most smartphones are embedded with multiple wireless interfaces, such as Wi-Fi, 2.5G/3G/4G, and Bluetooth. Wireless networks exhibit several different characteristics from wired networks such as narrower and varying capacity, disconnection, and random packet loss. (2) The small form factor of smartphones imposes constraints on the user interface of mobile apps. For example, manipulation through a touch screen is not as accurate as that through a mouse. A smartphone keyboard, both physical and on-screen, is not large enough to fit all keys of a standard-size keyboard. The small screen size of three to four inches limits the amount of information that can be shown by the application. To address the aforementioned issues, a mobile app typically has a different user interface from its PC counterpart and needs to be redesigned. (3) Smartphones provide limited computation and battery resources. Compared to PC, smartphones have lower computation resources such as CPU, memory, and storage. More importantly, the dependence on battery power poses a fundamental constraint on the available computation and network resources in a smartphone. (4) Unlike the consumer PC market which is dominated by Microsoft Windows, the smartphones today have diverse operating systems: Google Android, Apple iOS, Symbian OS, BlackBerry OS, Microsoft Windows CE (including Windows Mobile and Windows Phone 7), Palm webOS, etc. Each OS platform is not compatible with the other, and thus each mobile app needs to be specifically developed for a target platform. Besides, smartphones are typically based on an instruction set architecture for embedded systems such as ARM, while PCs are based on a regular architecture such as x86.

The unique characteristics of smartphones have given rise two fundamental problems in the emerging mobile computing platform: how to mobilize applications and

how to effectively deliver them. *Application mobilization* means the process of enabling a PC application to be used from mobile phones, specifically for smartphones in the context of the proposed research. Although the smartphone platform is capable of supporting PC-grade applications, the heterogeneity between the PC platform and the smartphone platform does not allow a PC application to directly run in a smartphone. Traditional methodology of application mobilization requires explicit development and has several limitations including time and cost requirement, usability, and feature parity. *Application delivery* means the delivery of application contents to the client via the networks. A mobile app, be it created with traditional application mobilization or not, relies on the wireless interfaces embedded in a smartphone to enable network access. In contract to the PC platform, the weak connectivity of the smartphone platform is characterized by its limited and varying capacity, random packet loss, and temporary disconnection. The issues of the weak connectivity, if left unaddressed, would have a profound impact on the application performance delivered to the smartphone users.

In this dissertation, we extract two networking problems from application mobilization and application delivery in the context of smartphones. First, in application delivery, we focus on the problem of *leveraging heterogeneous interfaces available at a smartphone*. Most smartphones today are equipped with multiple and heterogeneous wireless interfaces. Approaches of leveraging multiple network interfaces fall under two broad categories: "one-interface at a time" approaches and the more recently studied "simultaneous use of multiple interfaces" approaches. For obvious reasons including the fact that the wireless interfaces are innately limited and heterogeneous in capabilities, the latter class of approaches have significant promise in improving the performance experience by the user. However, many of the approaches that fall into this category have focused on what can be defined as *bandwidth aggregation* as the primary goal to achieve. In other words, if there are two interfaces I_1 and I_2 available

with respective bandwidths of B_1 and B_2, the approaches focus on delivering the aggregate bandwidth of $B_1 + B_2$ to the user. We call the functionality provided by such approaches *simple aggregation* and argue that for most practical environments simple aggregation will provide no meaningful benefits to the user, because of the large degree of capacity heterogeneity among wireless interfaces. Consider, for example, a smartphone that supports both 3G and Wi-Fi data interfaces. 3G data rates support bandwidths of up to 100-500Kbps while Wi-Fi interfaces support 2-54Mbps. A simple aggregation of the bandwidths provided by the two interfaces will provide negligible improvement in terms of performance perceived by the user, with respect to a best-available-interface solution. Thus, an interesting and relevant question that arises in the context of application delivery for smartphones is the following: *Is there a more intelligent strategy for aggregation of multiple heterogeneous interfaces that will enable us to achieve "more than the sum of the parts" in terms of network performance?*

In the context of application mobilization, we focus on a networking problem of *remote computing from heterogeneous devices*. Remote computing provides a novel approach of application mobilization by running a PC version of the application on a PC and providing a remote view into that application on the smartphone. This approach offers several attractive benefits including zero-code mobilization, full-functionality applications, and seamless manageability. Besides application mobilization, remote computing itself is an important networking problem with several applications ranging from remote access of files and data, to access to applications installed on servers, to remote IT support. The more recent emergence of virtual desktop infrastructures (VDIs) that almost exclusively rely on remote computing software for access has further elevated the importance of the latter. The applications of remote computing and the proliferation of mobile devices have driven the need of remote computing to PCs from heterogeneous devices including smartphones and tablets. For examples, industry solutions of remote computing for mobile devices include Microsoft Remote

Desktop Mobile, VMware View Mobile, and Citrix Receiver Mobile. However, relying on remote computing to access applications from heterogeneous devices raises a major challenge since *remote computing protocols typically assume homogeneous platforms on either end of the remote session.* As a result, application views that are originally developed for PCs will be presented as-is on the smartphones. Several unique characteristics of smartphones make the PC view quite cumbersome to use - (i) The bounding box of a PC application is typically much larger than the screen real estate on a smartphone. This raises extra burden in zooming and panning in accessing UI elements in the PC application; (ii) Application interfaces on smartphones are typically layered to adapt navigation and information organization to the constraint of the screen size. PC application with a flatter structure may cause extra burden on the smartphone user; (iii) Independent of the above issues that increase user effort, performing the same number of actions on the smartphone as on the PC is also subjectively burdensome to the user due to the constrained environment. (iv) The back and forth exchange of data between the server and client that could impose data usage burdens on the wireless link. To address the above issues, the second question we ask in this dissertation is: *Is there a more intelligent design of remote computing protocols that achieves better user performance by delivering suitable application views to heterogeneous devices?*

In this dissertation, we focus on protocol design that addresses the two networking problems identified in the context of application delivery and application mobilization: aggregating heterogeneous wireless interfaces and remote computing from heterogeneous devices. To provide practical solutions that achieves rapid application delivery and mobilization, we consider system implementation of the protocols that requires just software modifications, since smartphone hardware typically has an integral design and is not extensible. Thus, the contributions of this work are threefold:

- First we identify interface aggregation and remote computing as protocol primitives and proposing novel approaches in adapting the primitives to solve the problems of application mobilization and delivery for smartphones. Specifically, we propose to leverage the interface heterogeneous when using the primitive of interface aggregation in the context of rapid application delivery for smartphones, and we propose to transform application view when using the primitive of remote computing in the context of rapid application mobilization for smartphones.

- Second, we develop principles and design elements in transport-layer and application-layer protocols respectively for aggregating heterogeneous interfaces and remote computing across heterogeneous devices. In transport layer, we propose a protocol called *super-aggregation* that effectively aggregates the heterogeneous interfaces available in smartphones and achieves performance beyond simple aggregation. *Super-aggregation* leverages the the heterogeneity that naturally exists among different wireless interfaces. With appropriate knowledge of the data being transferred over the interfaces, *super-aggregation* is able to achieve aggregate throughput that is better than the sum of the throughputs achievable in each of the interfaces individually. In application layer, we propose a mobile remote computing protocol for heterogeneous devices called *MORPH*. *MORPH* is built atop of traditional remote computing and enables application transformation that dynamically transforms the PC view into one that is suitable for heterogeneous devices including smartphones and tablets.

- Third, we develop full-fledged experimental systems for application mobilization and delivery that use the respective protocols. The *super-aggregation* system is designed as a network optimization middleware that requires only client-side software modifications. To achieve rapid application mobilization, we propose a

system solution called **Mobile* that is based on the *MORPH* protocol. **Mobile* uses a proxy-based network architecture to mobilize PC applications for smartphones without explicit effort in development. It enhances the usability of the mobilized applications with a new mobile computing paradigm called *programming by transformation* enabled by the *MORPH* protocol. Finally, we propose the integrated operation of *super-aggregation* and *MORPH* to achieve a complete solution for rapid application mobilization and delivery for smartphones.

Thus, the central thesis of this work is that remote computing and interface aggregation, with appropriate adaptations, can be used to respectively achieve rapid application mobilization and application delivery for smartphones. The adaptations constitute the core research contributions and consist of the MORPH remote computing protocol and the super-aggregation solution for interface aggregation.

The rest of the dissertation is organized as follows. In Chapter 2, we provide the origin and history of the problem, where we discuss related work in both academic literature and industrial solutions. In Chapter 3, we present the protocol and solution of *super-aggregation* that achieves rapid application delivery by effectively aggregating heterogeneous interfaces available in smartphones. In Chapter 4, we present the design of the *MORPH* protocol that enhances the performance of remote computing for heterogeneous devices with view virtualization and transformation services. In Chapter 5, we present the design and evaluation of $MORPH_{Aggregation}$, the aggregation transformation service that reduces time and effort in accomplishing tasks in remote computing from smartphones. In Chapter 6, we present the design of translation service and traffic suppression service of *MORPH*, and we present **Mobile* as the rapid application mobilization solution based on *MORPH*. In Chapter 7, we present the design of other transformation services of *MORPH*, and we present the integrated evaluation of *super-aggregation* and *MORPH*. In Chapter 8, we present the conclusion and discuss future research that can potentially be spawned from this dissertation.

CHAPTER II

ORIGIN AND HISTORY OF THE PROBLEM

Smartphones are an emerging class of mobile phones that offers advanced computation capabilities and network connectivity. They enable mobile applications, which are often called mobile apps, by providing a complete operating system and a full-fledged application platform. With the rapid growth in industry, smartphones have recently hit a long-anticipated milestone: they overtook PCs in terms of the global shipments [22]. The dramatic growth in smartphones, e.g. 87.2% in the worldwide market in 2010 [10], demonstrates its promising potentials. Smartphones have occupied more than one quarter of the mobile phone population in US and Europe [7], and they are expected to dominate the mobile market in the near future [23]. The technology advances in the smartphones have made them a promising platform for mobile computing that has drew attentions from both the research community and the industry. Apple's App Store, the largest digital distribution platform for mobile apps, provides a collection of 350,000 mobile apps [4], while Android is catching up with 300,000 apps [34]. In this chapter, we discuss work in literature and industry solutions related to application mobilization and delivery for smartphones.

2.1 Performance enhancement with multiple interfaces

There has been significant work in the context of enhancing network performance using multiple interfaces, and approaches have tackled the problem fall under two broad categories: "one interface at a time" approaches and the more recently studied "simultaneous use of multiple interfaces" approaches. The former approach aims to select the best network path out of the available interfaces [46]. Since wireless networks are innately limited and heterogeneous in nature, the latter approach is more

promising in improving the performance experienced by the user. Most of the works in this category have focused on achieving bandwidth aggregation of multiple interfaces. pTCP [51], WAMP [75], RMTP [62], MC^2 [73], MAR [72], LS-SCTP [30], SCTP-CMT [53] perform bandwidth aggregation at transport layer, while [68] achieves bandwidth aggregation at network layer. R^{2}CP [50] and PRISM [54] also include mechanisms to address issues in wireless network, but they both require end-to-end deployment. They all provide linear throughput improvement by simply dividing traffic to multiple interfaces.

Heterogeneous interfaces of smartphones have also been used to improve other performance metrics other than throughput. CoolSpots [67] uses Bluetooth-enabled access points to reduce energy consumption on wireless divides with both Wi-Fi and Bluetooth interfaces. Cell2Notify [31] uses cellular radio to wake up Wi-Fi interface on a smartphone when there is incoming traffic. It improves battery lifetime of VoIP over Wi-Fi by reduce energy consumption in idle time of Wi-Fi. Context-for-Wireless [70] proposes an adaptive radio selection strategy between Wi-Fi and cellular based on context.

2.2 Transport-layer protocol design for wireless networks

A number of works have been proposed to extend or modify existing transport-layer protocols, such as TCP, for wireless networks. They typically consider just one wireless technology. For example, TCP-ELN [35], WTCP [74], and STP [48] are TCP-based protocols for WLAN, WWAN, and satellite network, respectively. On the other hand, they require modification in entities beside the mobile client, such as the access point and/or the correspondent host in the Internet. For example, Snoop [36] and WTCP [74] both address random wireless losses, but they are deployed at AP or both ends, respectively. Freeze-TCP [47] also uses flow-control mechanisms to freeze

TCP connection before blackout happens. However, it requires appropriate TCP implementation on sender side and accurate prediction of blackout occurrence time to make the mechanism work.

2.3 Application Mobilization

Application mobilization is done today using one of three different strategies: (i) *homegrown solutions* where enterprises directly invest resources in developing custom mobilized applications; (ii) *third-party solutions* where application vendors such as SAP, Oracle and Microsoft provide mobilization platforms that can in-turn be used by enterprises to mobilize applications with appropriate configurations; and (iii) *cross-platform solutions* like web applications that are compatible with multiple platforms including smartphones. The above methodologies to mobilization however come with several limitations. (1) Traditional application mobilization is a time and cost consuming process since all (or partial) functionality of a PC application needs to be rewritten for the smartphone platform. The mobilization time can vary depending upon the complexity of the application and can range from a few months to over a year. The diversity among smartphone platforms requires separate development effort for each platform, and that further magnifies the time and cost requirement in mobilization. (2) While the user interface of PC applications is designed for PC interface, it does not fit with the smartphone interface that is more constrained and is different from that of PCs. User interface needs to be adapted in the application mobilization process to cope with the constraints in the smartphone interface, otherwise the usability of the mobilized application would be impaired and the user would be frustrated. (3) Another important consideration is what the feature parity of the mobilized application will be. While some popular PC applications such as office suite have been mobilized by application vendors, the mobile versions typically expose only a subset of the capabilities of the full-blown PC applications.

2.4 Remote computing

There have been two types of remote computing. The first approach is where information is passed between the server and the client in the form of raw pixel data. The VNC [71] remote desktop application is an example. The VNC server encodes the pixel data of the remote computer and sends the encoded bit stream to the VNC client, which decodes the bit stream and the renders the screen display on the local computer. To save the amount of data transferred, the VNC server periodically polls the pixel data of the full screen of the remote computer to detect the regions that are updated and sends only the changed portion of the screen instead of continuously sending the full screen frames of the remote computer. VNC is a cross-platform solution and can work across multiple operating systems for both the remote and local computers. Examples of solutions based on VNC are UltraVNC and RealVNC [71]. The second type is based on graphical primitives, which are basic drawing commands provided by the operating system. RDP [21] is a Microsoft application and protocol that falls under this category. MobiDesk [37] proposes a thin client solution for mobile devices by optimizing the WAN traffic involved in performing remote computing. The solution is primarily meant for mobile laptops and is similar in principle to other remote computing approaches. PCoIP [20] is another product that optimizes remote computing traffic, especially video over IP networks. It improves user experience by adapting video quality to the network conditions in remote computing.

Remote computing solutions has also been made available to smartphones. There has been an open source thin client called AndroidVNC [1] available in Google Android phones. There are also commercial remote computing solutions available to different smartphones phones including iPhone, Android, and BlackBerry: MochaSoft RDP and VNC clients[19], iTeleport [12], TeamViewer [25], and LogMeIn ignition [15].

2.5 Macros

The notion of operator aggregation is related to the concept of macro, which is a sequence of instructions that has been recorded and can be replayed by the user. There have been two types of macros proposed in the literature, and they have different limitations.

The first type is application macros provided by the application developers in certain application software. One of the most popular application macros is Microsoft Excel macro [18]. Excel allows users to record their operators in the form of a Visual Basic script. The other example is iMacros [11], which is a browser plug-in that allows users to record their operators when browsing the web. However, each application macro only works for the specific application. The user has to rely on the application developer to provide such functionalities. The scope of the application macros is also limited. An application macro cannot work across multiple applications. Some functions of an application may not be captured by the macro system. For example, iMacros does not record the operator of printing a web page.

The second type is raw macros, which records and replays the raw activities such as mouse clicks at a certain coordinate and keystrokes. An example of raw macro systems is AutoHotkey [5]. Although raw macros work for generic applications in PC, it cannot replay the intended tasks robustly. Since raw macros are defined by the raw system variables, it would fail replay the recorded task if the system environment is not the same as the recorded state. For example, moving/resizing the window would cause the mouse click at the same coordinate to activate a different function. Raw macros cannot respond to adaptive user interfaces, such as the truncated menu that shows frequent items in Microsoft Office and other applications. The timing of replaying the macro is a big headache to raw macros, since the appearance next function may be delayed due to the current computation loading, and raw macros have no information regarding the application context. The concept of raw macros has been studied in

the area of Programming by Demonstration (PbD) [43, 60, 76, 52, 55], where most solutions are designed for PC users instead of smartphone users.

Macros have also been applied in the context of network systems and mobile computing. In [42], the authors present a raw macro-based solution for remote computing. Mugshot [63], CoScriptor [59], SmartBookmarks [52], and Chickenfoot [40] are application macro solutions for web applications. WikiDo [56] is crowd sourcing based IT-support solution for allowing non-technical users to help each other in setting up/using IT applications. The solution records operators of one user on one computer and replays the recorded operators on another computer. While the concept of application agnostic macros proposed in WikiDo is similar to our solution with respect to the UI automation framework used, our solution is specifically designed for accessing macros from a smartphone, and it provides features for macro extensibility, exception handling and a macro suggestion tool. Apple automator [6] is an application for MAC OS that allows users to define workflows to access a user recorded set of functions automatically. The user is allowed to choose from a limited set of functions from each application to be used for the workflow. Tasker [24] is a similar automation application for the Android OS for automating a pre-defined set of operations on different apps running on the Android phone.

2.6 Mobile Application

UI customization has been applied in the context of smartphones to realize mobile-friendly interfaces. Merlion [64] is a solution for creating application mash-ups that allow users to define a smaller subset of UI elements to be visible when using the application remotely from a smartphone. The solution requires a design phase to customize the mobile app interface, and the frontend purely relies on VNC to display the selected UI elements. In [58], the authors propose a more reliable UI mash-up solution by performing image recognition of the UI elements accessed. Again the

users can manually select a modified simple interface for mobile UI for the application they want to access. However, this approach requires a complex image recognition component to identify the user intent and hence all the possible UI elements should be identified statically. PageTailor [39] introduces reusable customization for mobile users, but the solution is specific to web pages instead of general applications.

A number of mobile application development frameworks such as Appcelerator [3], appMobi [2] provide solutions for write-once-deploy-everywhere mobile apps using a single API framework. However, these solutions require a manual re-write of applications involving significant developer effort.

Content adaptation is a technology that is typically used to transform web content to adapt the device constraints of mobile phones. In [77], the important content of a web site is re-arranged or made bigger to improve the readability in a small screen. FeedCircuit [8] is a technology used by Google to adapt RSS feeds for the mobile browser in Android phones. Highlight [66] and Flashproxy [65] are solutions that allow users to access web applications from mobile phones that lack certain capabilities, such as JavaScript and Flash. All the above content adaptation solutions are designed for static web applications instead of native PC applications or rich web applications.

CHAPTER III

ON EFFECTIVELY EXPLOITING MULTIPLE WIRELESS INTERFACES IN MOBILE HOSTS

3.1 Overview

In this chapter, we focus on rapid application delivery for smartphones that effectively leveraging their available wireless interfaces. Mobile devices, be it smartphones, tablets, or laptops, have gone through a sea-change in their capabilities over the last decade. One of the different dimensions along which they have evolved is that of connectivity. To cater to the requirement of mobile users of ubiquitous and best possible connectivity, most mobile devices today are equipped with multiple and heterogeneous wireless interfaces. For example, popular smartphones today ranging from the iPhone to the Google Android phone to the Blackberry all come equipped with multiple wireless data interfaces, including Wi-Fi, 2.5G/3G/4G, and Bluetooth. Not surprisingly, an interesting and relevant question that arises in the context of such mobile devices is the following: *What is the best approach to leverage the multiple interfaces available at a mobile device in terms of the performance delivered to the user?* In answering the question we argue that simple "bandwidth aggregation" approaches do not provide any meaningful benefits when the multiple interfaces used have highly disparate bandwidths as is true in many practical environments. We then present *super-aggregation*, a set of mechanisms that in tandem use the multiple interfaces intelligently and in the process is able to achieve a performance that is "better than the sum of throughputs" achievable through each of the interfaces individually. We prototype super-aggregation on both a laptop and the Google Android mobile phone and demonstrate the significant (up to 3x throughput) performance improvements it

provides in real-world experiments. We conduct both theoretical analysis and extensive experiments show that *super-aggregation* is able to improve throughput beyond the sum of the parts under most of the cases.

3.2 Super-Aggregation

3.2.1 Scope

The scope of this work is restricted to devices with multiple wireless interfaces. While several of the principles presented may be relevant for devices for multiple wired interfaces or heterogeneous wired and wireless interfaces as well, we do not delve into such scenarios in the thesis. The devices themselves can be either mobile computing devices such as laptops and mobile smart phones that have data capabilities. In terms of the wireless interfaces, the principles and solutions presented are agnostic to the specific technologies used.

Without impacting the generality of the proposed solutions, we rely entirely on a laptop and a mobile phone (Google Android) as the devices in the experimental set-up used for performance characterization, evaluation and proof-of-concept demonstration. Similarly, we use 3G and Wi-Fi as the heterogeneous wireless interfaces in all our experiments. The laptop is equipped with an Atheros 802.11b/g card and a 3G (EVDO) USB stick. The Google Android phone has built-in 802.11g and 3G (HSDPA) interfaces. Further details of the test-bed can be found in Section 3.7.

While the super-aggregation principles presented are extensible to other applications and protocols, *we ground most of our initial discussions of super-aggregation in the context of TCP acceleration.* We later revisit the extensibility of super-aggregation to both other applications/protocols and to environments with more wireless interfaces later in the chapter.

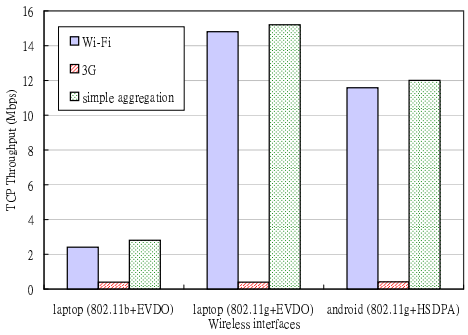


Figure 1: Marginal improvement of simple aggregation on heterogeneous interfaces

3.2.2 Problem Motivation

In this section we briefly illustrate the limitation of switching and simple aggregation and thus motivate the need for *super-aggregation*. Figure 1 shows measurement of TCP throughput as experienced by a multi-interface laptop that has 3G (EVDO) and Wi-Fi (g/b) connectivity. We emulate a perfect simple aggregation mechanism by opening two independent TCP connections through the two available interfaces, thus removing any synchronization bottlenecks. Hence, the result presented for simple aggregation is idealized.

The average throughput on the 3G link is much smaller than that on the Wi-Fi link and hence simple aggregation only gives marginal improvement: merely 3% in the 'g' mode and 16% in the 'b' mode. The measurement shows that simple aggregation is not an effective way of aggregating heterogeneous wireless interfaces in typical scenarios. The above result, while obvious, serves as the motivation for our investigation into super-aggregation principles that more intelligently leverage the multiple interfaces to achieve a throughput performance that is *better than the sum of the parts*.

3.2.3 Goals

The goals of this work, beyond the identification and development of super-aggregation principles, are as follows:

- **Deployability:** We believe that any aggregation solution for multi-homed mobile devices will have a significantly better adoption rate if it requires only mobile device side changes. Hence, one of our key goals will be to develop and realize super-aggregation through changes only at the mobile device.

- **TCP and beyond:** Since TCP is by far the most dominantly used transport layer protocol in today's Internet, we focus our goals almost entirely on TCP acceleration as the first application of super-aggregation. We however will revisit extending super-aggregation to other applications/protocols later in the chapter.

- **Prototyping:** Another key goal for the work is prototyping and demonstrating super-aggregation on real-world platforms including a laptop and a mobile phone.

3.3 Super-Aggregation Principles

In this section we present the different super-aggregation principles, and ground the discussions specifically in the context of TCP acceleration. At a high level, when aggregating a high-bandwidth interface with a low-bandwidth interface, a simple bandwidth aggregation strategy does not yield any significant improvements in performance. The super-aggregation principles on the other hand *explicitly leverage properties of the low-bandwidth interface that may be superior to those of the high-bandwidth interface to relieve any bottlenecks that prevent the effective utilization of the high-bandwidth interface.*

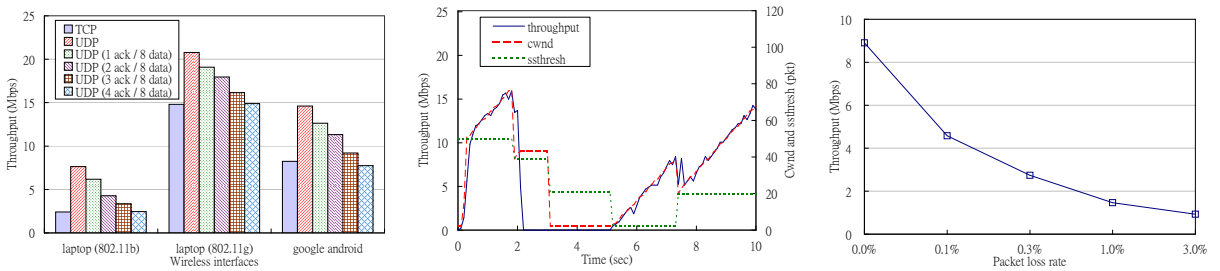
For each of the principles, we identify the rationale and present the high-level design. We arrive at the rationale by appropriately identifying an existing bottleneck in TCP's operation and show how super-aggregation may be used to relieve that bottleneck. For all the experiments we use the set-up discussed in Section 3.7 where both the laptop and the smartphone are connected to the backbone using a 3G link

and an Wi-Fi link. A file server hosted in a major university is used as the fixed point from where content is downloaded or to where content is uploaded. We revisit the solution details including the generalization of the principles to beyond TCP in the next section. Also, we present the super-aggregation techniques for TCP acceleration in the form of a layer 3.5 solution in the protocol stack. For brevity, in the rest of the discussions we refer to the high bandwidth interface as the Wi-Fi interface and the low bandwidth interface as the 3G interface.

3.3.1 Selective Offloading - Tackling TCP Self Contention

TCP Self Contention: Figure 2(a) shows the TCP throughput measurements for downstream traffic as observed at both the laptop and the smartphone. The figure also shows the throughput enjoyed by an application using UDP in a similar set-up. The UDP packet sizes are set to be the same as the TCP maximum segment size (MSS). The UDP throughput is 30% higher than the TCP throughput in the 802.11g network and as much as 70% higher in the 802.11b network.

Hence, the difference in performance is attributable to two possible factors: (a) upstream load imposed by TCP's ACK traffic and the resulting self-contention; and (b) TCP's congestion control algorithm potentially inhibiting the connection throughput. However, the experimental set-up is such that there is no contending traffic on the wireless legs, thus leading to the first factor as the dominating one. We refer to this cause as simply *ACK related self contention*. Although a typical TCP implementation sends an ACK for every two data packets, the ACK is significantly smaller with only a 20 byte IP payload as opposed to a 1480 byte IP payload for the data. *However, due to the overheads imposed by the 802.11 protocol, even small sized ACK frames end up contending on an equal footing to the data frames at the MAC layer.* This is because of both the byte overheads due to link layer headers and the preamble and other MAC operations such as inter-frame spacings and transmission of certain



(a) Impact from self-contention (b) Impact from blackout (c) Effect of random wireless losses

Figure 2: Motivation of *super-aggregation* for TCP

portions of the frames at base transmission rate.

We verify this self-contention hypothesis by explicitly sending bidirectional UDP traffic in the Wi-Fi network to mimic the behavior of TCP. We send 1464-byte UDP datagrams on the downlink, and 32-byte UDP datagrams on the uplink. As shown in Figure 2(a), we send one up to four ACKs per eight segments. The last case mimics self-contention of TCP ACKs, which reduces throughput from 20 Mbps to 15 Mbps in the 802.11g network. This is despite the fact that TCP ACKs take only 164 kbps. Other cases show that intermediate levels of self-contention also cause corresponding throughput reduction, and that TCP throughput can be increased by removing ACKs from the Wi-Fi network.

Selective Offloading: In this context, we propose an *offloading-ACK* mechanism to address self-contention in Wi-Fi networks. The key idea is to divert uplink ACKs to the 3G uplink to prevent them from contending with the downlink data in the Wi-Fi network[1].

There are, however, two challenges that need to be addressed in order for *offloading-ACK* to be viable in a real environment: (i) The 3G uplink may not have sufficient bandwidth to send the required number of ACKs that will sustain the maximum TCP downlink throughput on the Wi-Fi network. The low uplink bandwidth has two impacts on TCP. ACKs may be dropped at the transmission buffer, which renders

[1]We address obvious issues such as impact of ingress filtering later in the chapter.

the TCP sender unable to increase its congestion window or have more bursty transmissions. (ii) The 3G link has a larger RTT, which increases the RTT observed by the TCP sender and hence slows down the growth rate of its congestion window and hence the overall throughput enjoyed by the TCP connection.

We address both the above challenges by adding intelligence to the offloading-ACK mechanism to better control *when the ACKs are offloaded* and *how many of the ACKs are offloaded*. First, offloading-ACK is performed only when the consequent RTT inflation does not adversely impact the growth of the congestion window. This occurs when the congestion window is large (and hence the connection throughput is less dependent on congestion window growth rate), which naturally is also when self-contention will be near its peak. A simple heuristic we use for the offload-ACK threshold is the *ssthresh* value that TCP uses in its congestion avoidance algorithm. The value is 20 segments in our prototyping. Secondly, offloading-ACK is performed only to that fraction of ACKs that are indeed sustainable by the low-bandwidth interface. The remaining ACKs are sent as-is through the default interface. ACKs in the beginning of a congestion window are preferably sent over the low-bandwidth interface as opposed to those toward the rear-end of the congestion window to offset the delay differences and also to mitigate any adverse impacts of out of order receipt of ACKs at the TCP sender[2].

We revisit other properties of the offloading-ACK mechanism and synergies with the other proposed mechanisms later in the chapter.

3.3.2 Proxying - Overcoming Impact of Blackouts on TCP

Impact of Blackouts on TCP: A blackout for a wireless link occurs when the wireless channel experiences severe fading or the client undergoes a layer two or layer three handoff. TCP's performance is severely impacted by such blackouts, especially

[2]Note that a TCP sender that receives spurious ACKs (with sequence number less than a previously received ACK sequence number) will simply ignore the spurious ACKs.

in vehicular Wi-Fi networks, as an experimental study shows average 75-second blackouts [41]. Figure 2(b) shows the state of a TCP connection during a blackout that occurs during a 2 second period. The TCP sender, unaware of the blackout, will lose all packets transmitted during the blackout and hence will experience a retransmission timeout. The sender subsequently will enter slow start and drop its congestion window to one. Since the TCP sender cannot know the exact time the blackout ends, it relies on retransmissions of the first segment in the congestion window followed by an exponential backoff in the RTO if no ACKs are received. It is very likely that when the blackout ends at the mobile device end, the TCP sender is still in the midst of waiting for the expiry of a now inflated retransmission timeout. This unnecessary idle period coupled with the TCP connection starting back from a congestion window of one and a very small *ssthresh* (due to the back to back timeouts) render the TCP connection crippled to a low throughput performance. In this particular example, the throughput of the TCP connection is roughly reduced by half because of the two second blackout. Note that any mechanism such as the mobile device gratuitously sending an ACK when the wireless interface comes out of blackout is unlikely to address all the above problems.

Proxying: In this context, we propose a super-aggregation technique called *proxying-blackout-freeze* in which the 3G link is used to notify the TCP sender about blackout events on the WI-Fi link. The notification is in the form of a zero-window advertisement as if it were sent from the receiver with the WI-Fi interface's IP address. The notification will freeze the TCP connection when blackout occurs. When the WI-Fi interface recovers from the blackout, a resume notification in the form of a non-zero window advertisement is sent through the WI-Fi interface. According to RFC 1122 [29], the TCP sender should enter persist mode upon receiving a zero-window advertisement. When it receives the window update at a later point it restarts sending segments from the first unacknowledged sequence number without

reducing the congestion window or the slow-start threshold.

An important challenge that needs to be addressed is the actual blackout detection that must be done in real time and ideally with a low overhead. In the proposed super-aggregation technique, we implement a hybrid blackout detection mechanism to achieve a high responsiveness with a low overhead. An active link probing is performed when a passive link monitoring mechanism indicates that a blackout has likely occurred. The passive link monitor merely tracks if any packet is seen at all on the wireless interface of interest. If no packets are seen for more than 200 ms, the wireless client sends an ICMP ping message to its default gateway on the wireless interface to verify if a blackout has actually occurred. The zero-window advertisement through the 3G interface is generated as soon as the blackout is verified. The monitor module continues sending of ICMP ping messages to the default gateway of the WI-Fi interface until it receives a response to its query. As soon as recovery from the blackout is verified (through the receipt of response to the ping) a window update is sent to the TCP sender to resume the connection.

Another challenge is that TCP sender may not implement zero-window behaviors suggested by RFC 1122. For example, Linux (kernel 2.6.27) TCP implementation responds to zero window advertisement only when there is no outstanding packets. This can be resolved by combining the mechanism in next subsection. New ACKs are sent via 3G interface to make TCP sender believe all outstanding packets are received, and use *mirroring-loss-fetching* to recover those packets without impacting the performance on Wi-Fi.

3.3.3 Mirroring - Hiding Random Losses from TCP

Random losses: TCP is well known to suffer in the presence of random losses in wireless environments. Figure 2(c) shows the TCP throughput when introducing random wireless losses into the network. Even a 1% packet loss can reduce a TCP

connection's throughput to less than 20% of the achievable performance. As identified by a multitude of prior works, the main cause of the degradation is TCP's interpretation of all packet losses to be due to network congestion and the consequent reduction of its sending rate by half. However, the above mentioned random wireless losses may occur due to a high bit error rate in the wireless network due to low signal-to-noise-ratios because of channel fading, large tx-rx distances, or interference.

Thus, if there is a reliable approach to distinguish congestion losses from random losses and have the TCP sender react only to congestion losses, considerable improvements can be achieved.

Mirroring: In this context we introduce a concept of *random-loss hiding* for TCP connections, wherein a loss classified as a random loss is not reported by the mobile device back to the TCP sender. If such losses are not reported (positive ACKs are sent as if such segments were received successfully) the TCP sender will not retransmit those segments thus compromising on the guaranteed reliability semantics. To facilitate such loss hiding without compromising on the reliability semantics the proposed super-aggregation technique establishes a *mirrored TCP connection* through the 3G interface with the goal of fetching only the segments lost due to random loss. Segments fetched using the mirrored connection are then inserted back into the byte stream of the original connection to fill the holes created by the random losses. Such loss hiding and lost segment fetching through the 3G interface can provide considerable benefits for the TCP connection established through the WI-Fi interface. However, several key challenges need to be tackled. We now briefly elaborate on the challenges and the solutions proposed.

Loss distinction: How the proposed technique performs successful distinction of random and congestion losses is an important design element. However, note that the link layer at the mobile device will *receive corrupted frames* that it will then discard due to the errors unlike segments lost due to congestion that it will not receive in

the first place. With an appropriate interface into the link layer the proposed super-aggregation technique gathers information on frames discarded due to corruption and consequently classifies losses as congestion losses or random wireless losses.

TCP connection mirroring: The mirroring of a TCP connection, in general, will require application layer knowledge. While performing such mirroring is one option, the proposed super-aggregation technique relies on a simple connection set-up replay for mirroring the TCP connection. In other words, the sequence of messages exchanged since the set-up of the original connection is replayed in order to mirror the original TCP connection. With typical Internet applications (HTTP, FTP, SMTP, CIFS, P2P) we have considered thus far such a replay suffices.

Selected and Fast fetching: Since the 3G link has a considerably lower magnitude in terms of data-rate, a brute-force simple fetching of all the content just to retrieve the randomly lost segments is clearly not a viable strategy. Hence, the proposed mirroring mechanism performs selected and fast fetching, whereby the receiver proactively acknowledges segments that it does not need irrespective of whether it was received or not. Ideally, such segments will be purged from the TCP sender's buffer even before being transmitted by the sender. The only sequence numbers that the receiver does not acknowledge unless received correctly are the sequence numbers corresponding to the randomly lost segments. We place a guard time before fetching the desired segment to make the space for it to squeeze the narrow 3G link. The guard time is at least one segment size divided by data rate, and is configured as 256 ms in prototyping. We later show in the performance evaluation section on how such a fast and selected fetching scheme performs effectively. Note that some newer implementations of the TCP sender perform an explicit check for an incoming ACK sequence number being smaller than the right edge of the current congestion window. However, even under such conditions proactive ACKs paced correctly can appropriately accelerate the progress of the mirrored connection. However, some application

level range-fetching (such as supported by HTTP and FTP) would be required to eliminate redundant transmission of content on the mirrored connection.

Sequence number offsets: Since every new TCP connection establishment results in a new start sequence number, the mirroring mechanism appropriately offsets the sequence number of segments (or bytes) received on the mirrored connection to retrieve the sequence number pertinent to the original TCP connection.

3.3.4 Integrated Operations

The three mechanisms proposed for TCP acceleration can seamlessly work with each other. Moreover the traffic processing sequence would be as follows: *mirroring-loss-fetching*, *offloading-ACK*, and *proxying-blackout-freeze*. When segments arriving at the Wi-Fi interface have holes that have been attributed to random wireless losses, *mirroring-loss-fetching* hides them and generates a positive ACK for the latest segment. The generated ACKs are sent via the 3G interface using *offloading-ACK* mechanism. Finally, *Mirroring-loss-fetching* also establishes a mirrored connection to recover the randomly lost segments, and shares the 3G uplink capacity with *offloading-ACK*. If no packets are received for a period of time on the WI-Fi interface, *proxying-blackout-freeze* detects the blackout and freezes the Wi-Fi TCP connection using a notification through the 3G interface.

Finally, each *super-aggregation* principle shown in this section does outperform simple aggregation in a setting with heterogeneous interfaces by leveraging the multiple interfaces intelligently. However, *super-aggregation* in principle may include as one of its mechanisms simple bandwidth aggregation as well especially when the interfaces have bandwidths of similar orders.

3.3.5 Super-Aggregation for Upstream Communication

Although the *super-aggregation* principles presented thus far were discussed in the context of downstream data transfer, they can be adapted for upstream data transfer

in a straightforward manner. *Offloading-ACK* is realized by sending data packets on the Wi-Fi interface but labeling the 3G interface address as the source IP. Although data is sent on the Wi-Fi interface, ACKs will be received on the 3G interface to avoid self-contention. *Proxying-blackout-freeze* is similar to the downstream case, but zero-window advertisement and window updates are sent to the TCP sender in the local machine. *Mirroring-loss-fetching* for upstream traffic doesn't require establishment of a real mirrored connection. It merely retransmits random losses and masks the loss information from the local TCP sender.

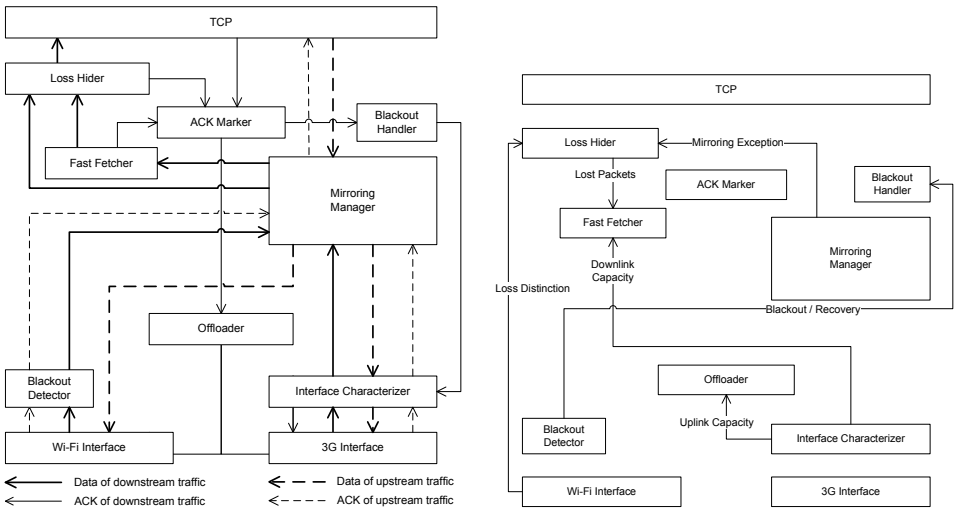
3.4 Super-Aggregation Architecture

3.4.1 Deployment Model

The *super-aggregation* principles presented in the last section can be implemented as a layer-3.5 software middleware in the mobile host. It can be implemented in the Linux kernel uses NetFilter to capture and process TCP packets traversing the network stack, or generate packets if necessary. The *super-aggregation* principles only require deployment at the mobile device and do not require any modification at the remote host or intermediate routers. The TCP implementations on the remote host and the mobile device are unaware of the *super-aggregation* principles that improve their performances transparently. With this deployment model, *super-aggregation* can enhance end-to-end performance of mobile host with any legacy TCP-based server.

3.4.2 Software Architecture

The implementation of each *super-aggregation* principle is divided into multiple components in the software architecture, as shown in Figure 3(a). We now briefly explain the components. *Offloading-ACK* is realized with the ACK Marker and Offloader components. *Proxying-blackout-freeze* is realized with the Blackout Detector and Blackout Handler. *Mirroring-loss-fetching* is realized with the Mirroring Manager, Loss Hider, and Fast Fetcher. Interface Characterizer is a common component used



(a) Data flow (b) Control flow

Figure 3: *Super-aggregation* architecture

by *offloading-ACK* and *mirroring-loss-fetching*.

Figure 3(a) shows packet flows across the *super-aggregation* components. There are four types of TCP packets in the diagram: data packets and ACK of downstream traffic; data packets and ACK of upstream traffic. The four types of packets are illustrated with different arrows in the packet flow diagram. For ease of exposition we describe the *super-aggregation* operations in the order of upstream data (request from client), upstream ACK, downstream data (content from server), and downstream ACK. Figure 3(b) shows the exchange of control messages among the *super-aggregation* components. The details are explained along with the data traffic.

Data of upstream traffic: Each packet generated by the TCP layer is recorded by the Mirroring Manager for establishing a mirroring connection on the 3G interface. The Mirroring Manager duplicates each data packet and puts the IP address of the 3G interface in the IP header. The data packets of the original connection are sent to the Wi-Fi interface. The duplicate packets are sent via the 3G interface in the same order to establish a mirroring connection and request for the same content from server.

ACK of upstream traffic: When a packet is received on the downlink of any

wireless interface, it could be an ACK for previously sent data packets or data packets sent by server in response to user request. Both data packets and ACK packets may belong to the original connection or the mirroring connection. All packets received on any wireless interface are sent to the Mirroring Manager, which differentiate the four cases. It distinguishes packets belonging to the original connection and the mirroring connection by comparing destination IP addresses with the record of mirroring. Data packets and ACKs are differentiated from each other by calculating their TCP payload length. TCP ACKs belonging to the original connection are sent to TCP layer directly without any modification. ACKs belonging to the mirroring connection are processed by Mirroring Manager itself. It retransmits data packets if they are lost in the Internet so that requests on the mirroring connection is always the same as the original one.

Data of downstream traffic: Similar to ACKs of upstream traffic, downstream data packets belonging to the original connection and the mirroring connection are treated separately. The Mirroring Manager first checks if data seen on the original connection and the mirroring connection are identical. If the data seen on both connections are different, mirroring cannot be supported since the TCP sender serves different data on the two paths. The Mirroring Manager falls back to no mirroring and sends a mirroring exception notification to the Loss Hider to stop hiding packet loss.When mirroring is enabled, the data packets belonging to the original connection are sent to the Loss Hider, and those of the mirroring connection are sent to the Fast Fetcher. Loss Hider sends all in-sequence data packets to TCP layer. If it finds a hole in the received data packets, it retrieves the loss information from the lower link layer. If the hole is attributed to a random wireless loss, the Loss Hider stores out-of-sequence packets in its buffer and notifies the Fast Fetcher about the lost packets to fetch on the mirrored connection.It sends a TCP ACK for highest data sequence number it has seen in order to hide the losses from the TCP sender. The Loss Hider waits for the Fast Fetcher to recover the lost packets and delivers all recovered packets

to TCP in sequence. Data packets coming from the mirroring connection are sent to the Fast Fetcher by the Mirroring Manager. The Fast Fetcher identifies the packets that were lost in the original connection and passes them to the Loss Hider. The Fast Fetching technique is used to control the process of sending ACKs in the mirrored connection to accelerate the retrieving of the lost packets.

ACK of downstream traffic: ACKs of downstream traffic are generated by the TCP layer and several components, such as the Loss Hider and the Fast Fetcher. All those ACKs are sent to the ACK marker to perform appropriate *offloading-ACK*. It marks all ACKs as offloadable so that the Offloader knows they can be offloaded to 3G interface. If an ACK only contains cumulative information and can be replaced by newer ACKs, ACK Marker adds a mark of replaceable. The Offloader takes ACKs of both the original connection and the mirroring connection for *offloading-ACK* via 3G interface. The Offloader gets the information of uplink capacity in the 3G interface from the interface characterizer. If the 3G interface has enough uplink capacity for offloading, it sends all the ACKs via the 3G interface. Otherwise, Offloader discards replaceable ACKs without affecting TCP operations.

Blackout handling: The Blackout Detector on the Wi-Fi interface monitors all activity events in the Wi-Fi network to detect blackout and link recovery events. It sends a probe message to default gateway if no activity is observed on the Wi-Fi interface for a period of time. When blackout is detected, the Blackout Detector sends a blackout notification to Blackout Freezer, which sends ACKs with zero-window advertisement to freeze the connection. When the link recovers from blackout, the Blackout Detector sends a link recovery notification to Blackout Freezer to resume the TCP transmission with a non-zero window update. The ACKs are generated with the latest ACK sent by the ACK Marker, and their flow window size is modified by the Blackout Detector. The ACKs directly go through the 3G interface since they are sent when the Wi-Fi interface is in blackout.

Interface characterizer: All four types of packets traverse Interface Characterizer if they are sent through the 3G interface. The Interface Characterizer measures capacity on both the uplink and the downlink in the 3G network. It calculates the remaining uplink/downlink capacity by subtracting data that have already been sent/received on the 3G interface. The uplink and downlink capacity information is sent to the Offloader and the Fast Fetcher whose decision relies on the remaining capacity. The Offloader reduces amount of ACKs sent to the 3G interface when there is not enough uplink capacity. The Fast Fetcher also adjusts the guard time that is inversely proportional to the downlink capacity on the 3G interface.

3.5 Super-Aggregation Beyond TCP

3.5.1 Generic Principles and Case Study

We now generalize the *super-aggregation* principles proposed in Section 3.3 and demonstrate their generic application with a case study of a non-TCP protocol. Each generic principle describes an approach of leveraging multiple interfaces of wireless devices that can increase throughput beyond the sum of the parts. We pick rate-adaptive video streaming [44] for the case study since it is a popular UDP application in the Internet. In such systems, the server sends video streams in the form of UDP datagrams to clients. To provide good video quality in response to capacity variation, the server adjusts its codec or sending rate based on available bandwidth to the client.

3.5.1.1 Selective offloading - tackling self-contention of client reports

Selective offloading chooses some packets to move from the Wi-Fi interface to the 3G interface. One design principle is moving some packets to the 3G interface to resolve self-contention in the Wi-Fi network, as in *offloading-ACK*. Moving small packets can provide significant improvements since overhead of sending them via the Wi-Fi interface is relatively higher. The other design principle is to use the 3G interface when overall performance is affected by some characteristics that the Wi-Fi

interface performs poorer than the 3G interface. In rate-adaptive video streaming, clients keep sending reports of traffic characteristics to the sender to rate adaptation. The rate adaptation and overall throughput may be impaired by intermittent availability of the Wi-Fi interface, while 3G interface provides much higher availability. A video client on mobile host is unable to send reports during blackout and handoff. The server may interpret missing of reports as client disconnection or network congestion. Incorrect characterization causes improper rate adaptation of the video stream. *Offloading-report* moves report packets to the 3G interface with high availability enables continuous reporting, which allows server to do timely and accurate rate adaptation. It also reduces packet loss rate of video on downlink, since small report packets cause self-contention in Wi-Fi networks.

3.5.1.2 Proxying - improving reliability and timeliness of command packets

Proxying improves performance of the connection on Wi-Fi by masquerading packets from 3G, which serves as a proxy when Wi-Fi is temporarily unavailable. One design principle is to enable communication when the Wi-Fi interface has blackouts, as in *proxying-blackout-freeze*. Adding control packets via the 3G interface can help in preventing blackout's adverse effects to application. The other design principle is to improve reliability and timeliness of some packets by sending them to both interfaces. Heterogeneous interfaces provide diversity in packet losses, so sending a redundant packet to the 3G interface can effectively improve end-to-end reliability. Video streaming clients send command packets to perform control operations, such as pause/resume video delivery and updating configurations. Losing command packets degrades response time perceived by the client. *Proxying-redundant-commands* sends a duplicate copy of those command packets to improve the reliability. It improves response time to the client and may also improve other dimensions of performance since commands are delivered more timely.

3.5.1.3 Mirroring - reducing loss rate of baseline frames

Mirroring creates a independent connection on the 3G interface, and same or related content is downloaded to improve performance by leveraging loss diversity. One design principle is decoupling some of high-layer mechanisms to the mirroring connections, as in *mirroring-loss-fetching*. This helps the client to separate operation of two mechanisms that have adverse interaction in Wi-Fi networks. The other design principle is reducing packet loss rate by fetching redundant contents from both connections, especially essential portions in the original connection. Scalable Video Coding [69] is commonly considered in rate-adaptive video streaming since it encodes video content into different quality levels in a scalable way. All clients receive baseline frames and those with higher capacity also receive enhancement frames that rely on baseline frames. Baseline frames are more critical packets and requires less bandwidth than enhancement frames. *Mirroring-baseline-frames* establishes a mirroring connection via the 3G interface and requests for a baseline video stream of the same video content. Having duplicate baseline frames from server can significantly improve overall video quality, especially in a lossy environment.

3.5.2 Generic Architecture

We present the generalized software architecture in a modular fashion such that it is evident how to reuse the common components for generic *super-aggregation* principles. As shown in Figure 3(a), some components in the *super-aggregation* architecture are specific for TCP, while others provide common functionalities needed in other principles. For example, Offloader is used for *offloading-ACK*, Report Offloading, and Voice Offloading. *Offloading-ACK* uses ACK marker to notify Offloader that ACK packets should be offloaded. The Report Offloading and Voice Offloading should have their own component to mark report packets and voice frames. The Offloader takes

all packets marked and split them according to available uplink bandwidth on 3G interface. Other common components include the Blackout Detector and the Interface Characterizer.

3.6 Theoretical Analysis

In this section we present an analytical model to study the throughput of *super-aggregation*. The analysis answers a fundamental question: *in what conditions does super-aggregation yield throughput more than sum of the parts?* The analytical model can also be used to assess the performance of *super-aggregation* when applied to other networks. Each of the three *super-aggregation* principles are analyzed separately in the corresponding network conditions. In the analysis of *offloading-ACK*, there is no blackout or random wireless loss. In the analysis of *proxying-blackout-freeze*, there are blackouts but no random wireless loss. In the analysis of *mirroring-loss-fetching*, there are random wireless losses but no blackout. For each principle, we first analyze the throughput of default TCP under the corresponding condition then the throughput of the *super-aggregation* principle. Throughout the analysis, we assume that the wireless link is the bottleneck of the end-to-end path. Table 1 lists the variables used in the analysis, and Table 2 lists the parameters and their values. At the end of this section, we presents the insights from the analysis to show that in most of the cases *super-aggregation* is able to provide throughput more than sum of the parts. The analytical model is validated with experiments in the next section.

3.6.1 Analysis of Offloading-ACK

When self-contention occurs to TCP, uplink ACK packets compete with downlink data packets for the same wireless network resources. A TCP receiver replies with an ACK for every two data segments. The saturated throughput of TCP at the wireless

Table 1: Variables in the *super-aggregation* analysis

Variable	Definition
$T_s(p)$	Saturated throughput of TCP protocol p (default TCP or *super-aggregation*)
$T(p)$	Average throughput of TCP protocol p (default TCP or *super-aggregation*)
t_{data}	Time to send a TCP data segment
t_{ack}	Time to send a TCP acknowledgement
$t_{frame}(l)$	Time to send a MAC frame of payload length l
$t_{mpdu}(l)$	Time to send a MPDU of payload length l
$t_{backoff}$	802.11 backoff duration
$t_{blackout}$	Duration of a blackout
$t_{interval}$	Interval between two blackout occurrences
p_c	Probability of contention
$bps(r)$	Bits per symbol of the 802.11 data rate r
W_s	Congestion control window size under saturation
$W(i)$	Congestion control window size at round i
W_max	Maximum congestion window size achieved upon packet loss
$N(p_l)$	Number of packets sent in a congestion avoidance phase
p_l	Packet loss rate
RTT_k	Round trip time on the path of link k (wifi or 3g)
RTO	TCP retransmission timeout
B	buffer size at the Wi-Fi AP

Table 2: Parameters in the *super-aggregation* analysis

Parameter	Definition	Value
h_{tcp}	TCP header length	32 bytes
mss	TCP maximum segment size	1448 bytes
h_{ip}	IP header length	20 bytes
h_{mac}	802.11 MAC header length	28 bytes
h_{mack}	802.11 MAC ACK length	14 bytes
t_{sifs}	802.11 SIFS	34μs
t_{difs}	802.11 DIFS	16μs
t_{slot}	802.11 slot duration	9μs
$CWmin$	Minimum contention window size	15
$t_{preamble}$	802.11 PLCP preamble duration	16μs
t_{signal}	802.11 PLCP signal field duration	16μs
h_{serv}	802.11 PLCP service field length	16 bits
h_{tail}	802.11 PLCP tail length	6 bits

link is determined by the time to send both the TCP data segments and ACKs.

$$T_s(tcp) = \frac{2mss}{2t_{data} + t_{ack}} \tag{1}$$

Either a TCP data segment or a TCP acknowledgement is sent as a frame in the Wi-Fi network, and each one subject to contention losses since the Wi-Fi AP and the Wi-Fi client are competing for channel access.

$$t_{data} = t_{frame}(h_{ip} + h_{tcp} + mss) \times \frac{1}{1 - p_c} \tag{2}$$

$$t_{ack} = t_{frame}(h_{ip} + h_{tcp}) \times \frac{1}{1 - p_c} \tag{3}$$

To simplify the analysis, we assume both the AP and the client stay with the minimum contention window.

$$p_c = \frac{1}{CW_{min} + 1} \tag{4}$$

$$t_{frame}(l) = t_{difs} + t_{backoff} + t_{mpdu}(h_{mac} + l) + t_{sifs} + t_{mpdu}(h_{mack}) \tag{5}$$

The expected duration in backoff also assumes minimum contention window.

$$t_{backoff} = \frac{1}{2} CW_{min} \times t_{slot} \tag{6}$$

$$t_{mpdu}(l) = t_{preamble} + t_{signal} + t_{symbol} \times \lceil \frac{h_{serv} + l + h_{tail}}{bps(r)} \rceil \tag{7}$$

Offloading-ACK eliminates self-contention of TCP in the Wi-Fi network by moving TCP ACKs to the 3G link. With the Wi-Fi link purely used for downstream TCP data segments, the saturated throughput does not contain time to send TCP ACKs.

$$T_s(tcp) = \frac{mss}{t_{data}} \tag{8}$$

Since there is no contention between the AP and the client, there is no overhead in retransmission due to contention losses.

$$t_{data} = t_{frame}(h_{ip} + h_{tcp} + mss) \tag{9}$$

TCP throughput can be further improved if the technique of frame bursting is enabled. Frame bursting is a technique supported by some commercial Wi-Fi APs, such as Linksys WAP55AG. It allows consecutive frames to be sent by the same host if no other host is competing for the channel access. With self-contention of TCP, most frames are not sent in a burst due to the contention. With *offloading-ACK*, frame burst works for most frames, and that reduces the time for sending a MAC frame since each MPDU is preceded by an SIFS instead of DIFS plus backoff.

$$t_{frame}(l) = t_{sifs} + t_{mpdu}(h_{mac} + l) + t_{sifs} + t_{mpdu}(h_{mack}) \tag{10}$$

The saturated throughput of both default TCP and *offloading-ACK* is derived above, but that does not translate into the average throughput observed in practice. The saturated throughput can be maintained only if the intermediate routers have enough buffer space the incorporate the natural fluctuation of congestion window size. In other words, the saturated throughput can be regarded as the upperbound of TCP throughput, which is achieved with sufficient buffer space in the Wi-Fi AP. The lowerbound of TCP throughput can be derived by assuming no buffer space in the Wi-Fi AP. That cause immediate packet drops if the TCP transmission rate is higher than the saturated throughput. The average throughput in such a scenario can be approximated as $\frac{3}{4}T_s(p)$ since the congestion window is cut down by half after a packet loss.

We provide a more detailed analysis based on the TCP congestion control mechanism. To derive the long-term average throughput, the analysis only considers the congestion avoidance phase of TCP. During congestion avoidance phase, congestion window is increased by one MSS every round, which length is the RTT. The congestion control window size (in the unit of TCP MSS) under saturation can be derived from the saturated throughput.

$$W_s = \frac{T_s \times RTT}{mss} \tag{11}$$

$$RTT = \begin{cases} RTT_{wifi} & \text{for default TCP} \\ \frac{RTT_{wifi}+RTT_{3g}}{2} & \text{for } \textit{offloading-ACK} \end{cases} \tag{12}$$

Let's assume at round r_s, TCP has congestion window equals to W_s. TCP keeps growing its congestion window beyond the saturated value. At round $r_s + i$, TCP has congestion window equals to $W_s + i$. It means that TCP sends i more packets than what the Wi-Fi link can absorb. In other words, i packets are accumulated in the buffer at the Wi-Fi AP at round i. Packet loss occurs at round $r_s + r_p$ such that

$$\sum_{i=1}^{r_p} i > B, \tag{13}$$

where B is buffer size at the Wi-Fi AP.

$$\frac{r_p^2}{2} + \frac{r_p}{2} - B > 0 \tag{14}$$

$$r_p = \lfloor -\frac{1}{2} + \sqrt{\frac{1}{4} + 2B} \rfloor + 1 \tag{15}$$

The maximum congestion window is achieved when the packet loss occurs.

$$\begin{aligned} W_{max} &= W(r_s + r_p) \\ &= W_s + r_p \end{aligned} \tag{16}$$

$$W_{min} = \lfloor \frac{W_{max}}{2} \rfloor \tag{17}$$

$$\begin{aligned} T(p) &= \frac{\frac{1}{2}(W_{max}+W_{min})\times(W_{max}-W_{min}+1)-1)\times mss}{(W_{max}-W_{min}+1)\times RTT} \\ &\cong \frac{3}{4}W_{max} \times \frac{mss}{RTT} \end{aligned} \tag{18}$$

3.6.2 Analysis of Proxying-blackout-freeze

To analyze the impact from blackout, we assume that blackouts occurred in the Wi-Fi network periodically. The period of a blackout is $t_{blackout}$, and the frequency of its occurrence is $f_{blackout}$. As shown in Figure 2(b), blackout causes an RTO timeout since the whole burst of packets are lost. When retransmission occurs after an RTO, if the

Wi-Fi network is still in blackout, the packet is retransmitted again with a doubled RTO. Let n be the number of retransmissions occurred in the blackout period.

$$t_{idle} = \sum_{i=0}^{n-1} RTO \times 2^i = RTO \times (2^n - 1) \geq t_{blackout} \tag{19}$$

$$n = \lfloor log_2(\frac{t_{blackout}}{RTO} + 1) \rfloor \tag{20}$$

$$t_{idle} = RTO \times (2^{\lfloor log_2(\frac{t_{blackout}}{RTO} + 1) \rfloor} - 1) \tag{21}$$

TCP goes back to slow start after the idle period. Since the slowstart threshold is also reduced to one MSS due to the repeated timeouts, TCP can only increase its congestion window by one MSS until it reaches the saturated congestion window. The time for TCP to resumes its congestion window from slow start back to the saturated value is:

$$t_{ss} = W_s \times RTT \tag{22}$$

After that, TCP enters its congestion avoidance phase and achieves throughput same as the the value derived in the previous subsection. It stays in congestion avoidance until next blackout occurs.

$$t_{ca} = t_{interval} - t_{idle} - t_{ss} \tag{23}$$

The overall throughput can be derived by averaging the amount of data transferred during the idle period, the slow start period, and the normal congestion avoidance period.

$$T(tcp) = \frac{t_{ss} \times \frac{(W_s+1)mss}{2RTT} + t_{ca} \times \frac{(W_{max}+W_{min})mss}{2RTT}}{t_{interval}} \tag{24}$$

Proxying-blackout-freezing can freeze the TCP transmission during the blackout and immediately resumes the transmission after the blackout. The TCP remains in

the normal throughput except during blackout.

$$t_{ca} = t_{interval} - t_{blackout} \tag{25}$$

$$T(proxying) = \frac{t_{ca} \times \frac{(W_{max}+W_{min})mss}{2RTT}}{t_{interval}} \tag{26}$$

3.6.3 Analysis of Mirroring-loss-fetching

To analyze the impacts from random wireless loss, we assume that each packet loss is an independent event with probability p_l. The analysis of TCP throughput is based on N, the number of packets sent in a congestion avoidance phase, which is defined as the duration between two packet losses. The probability distribution of N is derived by considering both congestion-related losses and random wireless losses.

$$Pr\{N = n\} = \begin{cases} (1-p_l)^{n-1}p_l & \text{if } 0 \leq n < N_{max} \\ (1-p_l)^{N_{max}} & \text{if } n = N_{max} \\ 0 & \text{otherwise} \end{cases} \tag{27}$$

$$N_{max} = \frac{1}{2}(W_{min} + W_{max})(W_{max} - W_{min} + 1) - 1 \tag{28}$$

N_{max} is the total number of packets that can be sent in a congestion avoidance phase when there is no packet loss. Since congestion-related packet loss occurs when the congestion window exceeds the maximum value derived above, N cannot exceed N_{max}. A congestion avoidance phase can be terminated earlier by a random wireless loss. Since each packet has a probability of p_l to be lost, the distribution of N under N_{max} is a geometric distribution.

$$\begin{aligned} E[N(p_l)] &= \sum_{n=0} N_{max} Pr\{N = n\} \times n \\ &= \sum_{n=0} N_{max} - 1(1-p_l)^{n-1} p_l \times n + (1-p_l)^{N_{max}} \times N_{max} \\ &= \frac{1-N_{max}(1-p_l)^{N_{max}-1}+(N_{max}-1)(1-p_l)^{N_{max}}}{p_l} + (1-p_l)^{N_{max}} N_{max} \\ &= \frac{1-(1-p_l)^{N_{max}}}{p_l} - p_l(1-p_l)^{N_{max}-1} N_{max} \end{aligned} \tag{29}$$

The derivation also demonstrates the qualitative characteristics of $E[N]$. When network capacity is high (N_{max} is large) or packet loss rate is high, the first component in $E[N]$ dominates, and thus the congestion window is limited by random wireless losses. When network capacity is low or packet loss rate is low, the second component in $E[N]$ dominates, and thus the congestion window is limited by congestion-related losses.

The expected upperbound of congestion control window size can be derived from the expected number of packets sent in a congestion avoidance phase.

$$E[N(p_l)] = \frac{(W_{max}(p_l)+W_{min}(p_l))(W_{max}(p_l)-W_{min}(p_l)+1)}{2} - 1 \tag{30}$$

$$\frac{3}{8} W_{max}(p_l)^2 + \frac{3}{4} W_{max}(p_l) - (1 + E[N(p_l)]) = 0 \tag{31}$$

$$W_{max}(p_l) = \sqrt{\frac{8}{3} E[N(p_l)] + 3} - 1 \tag{32}$$

$$W_{min}(p_l) = \lfloor \frac{1}{2} W_{max}(p_l) \rfloor \tag{33}$$

TCP throughput under packet loss rate p_l can be derived as follows:

$$\begin{aligned} T(tcp, p_l) &= \frac{E[N(p_l)] \times mss}{(W_{max}(p_l)-W_{min}(p_l)+1) \times RTT} \\ &\cong \frac{3}{4} W_{max}(p_l) \times \frac{mss}{RTT} \end{aligned} \tag{34}$$

Mirroring-loss-fetching prevents TCP from unnecessary reduction in congestion window by hiding random wireless losses to the TCP sender. The throughput achieved

with *mirroring-loss-fetching* is constrained by two factors. The first one is the throughput of data transfer in the primary connection, and the second one is the rate in recovering lost packets in the mirroring connection. In the mirroring connection, there is a guard time $t_{g,ack}$ before generating an ACK, and a larger guard time $t_{g,data}$ is applied when the ACK is expected to trigger a data segment that was lost in the primary connection. The value of $t_{g,ack}$ and $t_{g,data}$ depends on the data rate in the 3G connection. The rate in recovering lost packets T_r is determined by the average time to trigger a data segment (no matter lost or not) from the TCP sender.

$$T_r = \frac{mss}{p_l \times t_{g,data} + (1 - p_l) \times t_{g,ack}} \tag{35}$$

The overall throughput is the minimum of the throughput in the primary connection and the rate in recovering lost packets. If the distinction between congestion losses and random losses is enabled, *mirroring-loss-fetching* allows TCP to achieve the throughput in the primary connection as if there is no random wireless loss.

$$T(mirroring, p_l) = min(T(tcp, 0), T_r) \tag{36}$$

3.6.4 Insights from the Analysis

In this subsection, we present the insights from the theoretical analysis on the throughput of *super-aggregation* under different conditions. We specifically look at an important aspect of *the degree of heterogeneity required in the capacity of the two interfaces to observe aggregate throughput more than sum of the parts.* The following observations will be verified with extensive experiments in the next section.

The throughput improvement from *offloading-ACK* and/or *proxying-blackout-freeze* is independent from the capacity of the secondary interface. As shown in the theoretical analysis, the throughput formulas of *offloading-ACK* (Eq. 18) and *proxying-blackout-freeze* (Eq. 26) do not contain the capacity of the secondary interface. Those two principles are able to achieve a saturated throughput higher than sum of the

parts no matter how small the capacity of the secondary interface is. While the secondary interface has higher capacity than necessary, *offloading-ACK* or *proxying-blackout-freeze* does not exhaust its capacity, and the remaining capacity can be utilized with simple aggregation (discussed in Section 3.8). In general, *offloading-ACK* and *proxying-blackout-freeze* can provide aggregate throughput more than the sum of the parts under any degree of capacity heterogeneity.

Contrary to the above two principles, the throughput improvement of *mirroring-loss-fetching* depends on the capacity of the secondary interface. Eq. 36, when capacity of the secondary interface is significantly lower than that of the primary interface, the overall throughput of *mirroring-loss-fetching* is dominated by T_r. In that case, the aggregate throughput depends on the capacity of the secondary interface, as shown in Eq. 35. That is because the data transport of *mirroring-loss-fetching* is stagnated by waiting for loss recovery in the secondary interface. The analytical model can also be used to predict if *mirroring-loss-fetching* is beneficial when the capacity in the secondary interface drops down. If the capacity in the secondary interface is insufficient to benefit the overall throughput, *super-aggregation* should not hide all packet loss in the primary connection.

3.7 Performance Evaluation

In this section, we describe our prototyping on real multi-interface wireless devices and their performance in experimental field trials. We present performance of all *super-aggregation* principles on laptop in detail. Results on Google Android phone are summarized since they demonstrate similar performance. The performance results measured in the experimental testbed are used to validate the theoretical analysis in the previous section.

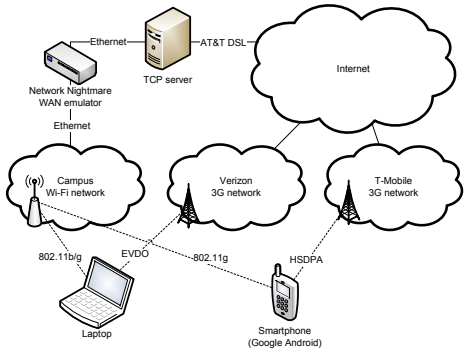


Figure 4: Topology of the *super-aggregation* experimental testbed

3.7.1 Experimental Testbed

Figure 4 shows the topology of the experimental testbed. It includes both typical types multi-interface wireless device: laptop and smartphone equipped with Wi-Fi and 3G. The laptop is equipped with an Atheros 802.11a/b/g PCMCIA card and a Verizon USB727 EVDO stick. Its operating system is Fedora 9 with Linux kernel 2.6.27. The driver of the Wi-Fi interface and the EVDO interface are MadWifi 0.9.4 and wvdial, respectively. The smartphone is T-Mobile G1, which has embedded Wi-Fi and HSDPA interfaces. It runs with Google Android operating system and Linux kernel 2.6.25. The testbed includes a TCP server to provide bulk data transfer for downstream throughput measurement and has no background traffic. Connections last for 100 seconds in each experiment. It is a desktop with Fedora 9 and Linux kernel 2.6.27.

To evaluate the performance of *super-aggregation* in real life, both mobile clients connects to the Internet via real service providers. The Wi-Fi access is provided by the campus network service at Georgia Tech. The 3G access on the laptop and the smartphone is provided by Verizon and T-Mobile, respectively. Since original RTT between Wi-Fi interface and the TCP server is too small, we add a WAN emulator between TCP server and campus network. We conduct RTT measurement from the Wi-Fi interface to 20 popular websites, and the average value is 58.6 msec. The

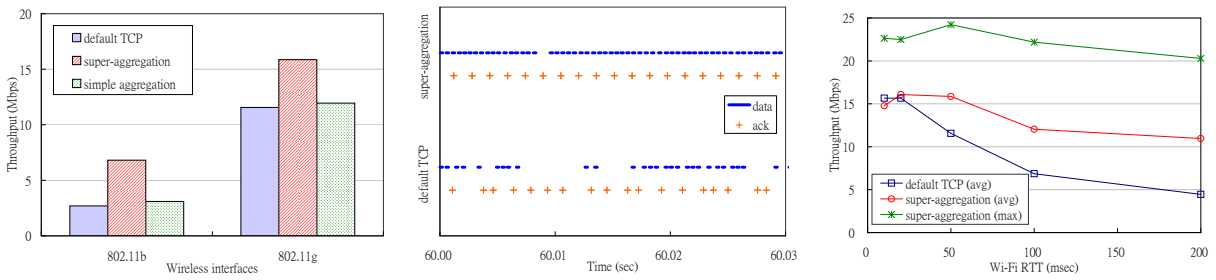
WAN emulator is configured with 50 msec RTT to make the testbed representative for multi-interface wireless devices.

Other than performance evaluation with real service providers, we also conduct more extensive experiments with diverse capacity in the two interfaces to validate the theoretical analysis. To manipulate the network capacity in the experiment testbed, we use a Wi-Fi AP (Linksys WAP55AG) and an emulated cellular network (Network Nightmare WAN Emulator). By using the network emulation, we covers a broad range of wireless technologies of heterogeneity capacity. The data rate in the Wi-Fi interface ranges from 6Mbps to 54Mbps. The data rate of the secondary interface also varies from 2.5G (GPRS/EDGE) with around 100kbps to 4G (WiMAX/LTE) with tens of Mbps.

3.7.2 Solution Prototyping

In order to evaluate *super-aggregation* performance with user-space implementation, we use a user-space TCP implementation so that *super-aggregation* can be realized below it. We select Atou (Almost TCP over UDP) [45], which is a validated tool for studying TCP performance. Both server and client program include bulk data transfer on top of Atou. Atou sends out segments and ACKs encapsulated in UDP datagrams, and TCP mechanisms are implemented based on standards. We use NewReno with SACK in Atou and verify that it gives almost the same performance as TCP (using iperf) as shown in the preliminary study in Section 3.3. Minor differences may appear between TCP and Atou for some scenarios. They may be caused by higher computation priority of TCP in kernel space or Atou's configurations not being exactly the same as that of a specific TCP implementation.

All three TCP-based *super-aggregation* principles are implemented in C and integrated with Atou. They are implemented only at the receiver-side source code and deployed to wireless clients. Sender-side codes are the original Atou, except some



(a) Throughput improvement on 802.11b/g (b) Avoiding self-contention in 802.11g (c) Effect of RTT on the Wi-Fi path

Figure 5: Performance of *offloading-ACK*

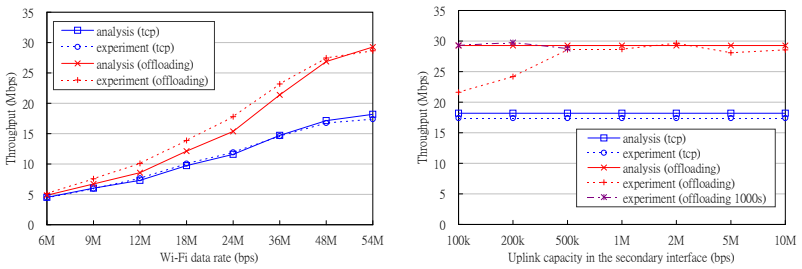
mechanisms are added for better TCP standard compliance (flow control and appropriate byte counting). We use gcc and ARM GNU/Linux cross-compiler to build the executable for laptop platform and Android platform, respectively.

3.7.3 Offloading-ACK Performance

In experiments, *Offloading-ACK* resolves self-contention and improves TCP throughput with same magnitude as expected from the motivation. It improves TCP throughput by 37% and 152%[3] on the laptop client using 802.11g and 802.11b, as shown in Figure 5(a). Figure 5(b) is packet traces captured with tcpdump. It demonstrates self-contention of default TCP, in which data packets and ACKs don't overlap in time. It also shows capability of *Offloading-ACK* to allow TCP to fully utilize the 802.11g interface for downlink data. Although average throughput of *offloading-ACK* is affected by RTT of the 3G path, it is still able to achieve maximum throughput during a connection, as shown in Figure 5(c). It gives higher improvement magnitude when RTT on Wi-Fi path is longer since relative impact from 3G interface's long RTT is smaller. The performance of integrated operations shows that *Mirroring-loss-fetching* can resolve the issue of long RTT on the 3G path.

Figure 6 compares the analysis and experimental results of both default TCP and

[3]To compare with TCP ACK aggregation techniques such as [33], we perform ACK aggregation (one for 12 packets) in 802.11g, and it only improves default TCP by 8.27% in experiments.



(a) Effect of Wi-Fi data rate

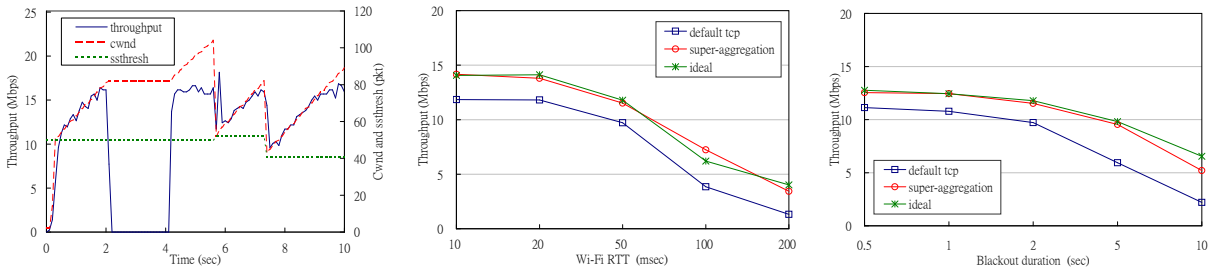
(b) Effect of capacity in secondary interface

Figure 6: Comparison of *offloading-ACK* analysis and experiments

offloading-ACK. The results show that the analytical model accurately capture the throughput performance of TCP and *offloading-ACK* under different circumstances. It is noteworthy in Figure 6(b) that when the cellular network has capacity lower than 500kbps, the observed throughput in our default experiment settings is lower than what expected by the analysis. The reason is because offloading ACKs through a cellular interface of narrow bandwidth slows down TCP in achieving the saturated throughput, even though the saturated throughput is independent of the cellular interface capacity. RFC 3465 [32] specifies that TCP should not increase its congestion window by more than two full segments with each acknowledgement, the growth rate in congestion control window is reduced when fewer acknowledgements are delivered. With the 100-second data transfer in default experiments, TCP with *offloading-ACK* does not have enough time to achieve the supported throughput. By extending the connection duration to 1000 seconds, the average throughput achieved in the experiments matches with the analysis.

3.7.4 Proxying-blackout-freeze Performance

Blackouts are generated every 20 seconds on average within the 100-second connection. Figure 7(a) demonstrates the effectiveness of *Proxying-blackout-freeze* by showing TCP states under blackout. TCP doesn't go to slow start or cut down congestion window when receiving zero-window advertisement via the EVDO interface. TCP



(a) Freezing TCP during blackout (b) Effect of RTT on Wi-Fi path (c) Effect of blackout duration

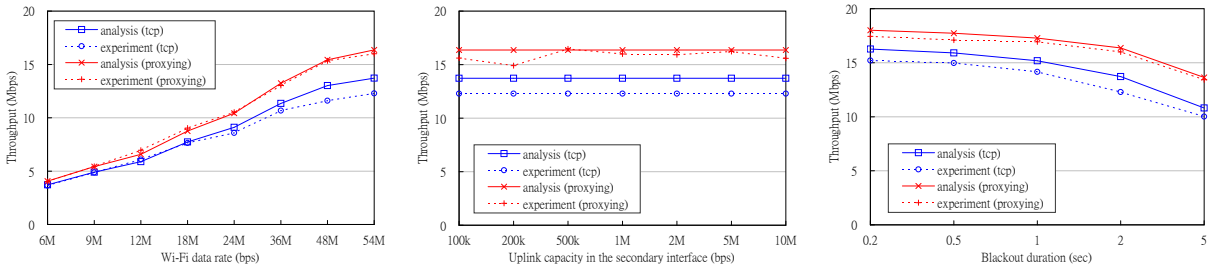
Figure 7: Performance of *proxying-blackout-freeze*

throughput is also immediately resumed after receiving window update after link comes back. Compared to default TCP in Figure 2(b), *proxying-blackout-freeze* improves TCP throughput by 87%. Figure 7(b) shows that *proxying-blackout-freeze* gives close-to-ideal throughput with different RTT on Wi-Fi path, where ideal throughput values are calculated as throughput with no blackout times the ratio of link availability. Improvement magnitude is 161% when Wi-Fi RTT is 200 msec since TCP spends more time to recover its congestion window. Figure 7(c) shows that *proxying-blackout-freeze* gives more improvement with longer blackout duration. It improves TCP throughput by 136% when average blackout duration is 10 seconds.

Figure 8 compares the analysis and experimental results of both default TCP and *proxying-blackout-freeze* under blackouts. The default blackout duration is 2 seconds. The analysis and the experimental results are very close to each other. The analysis of default TCP has higher throughput than experiments. That can be caused by the overhead in TCP retransmission in practice, such as a coarse-grained timer for TCP timeout.

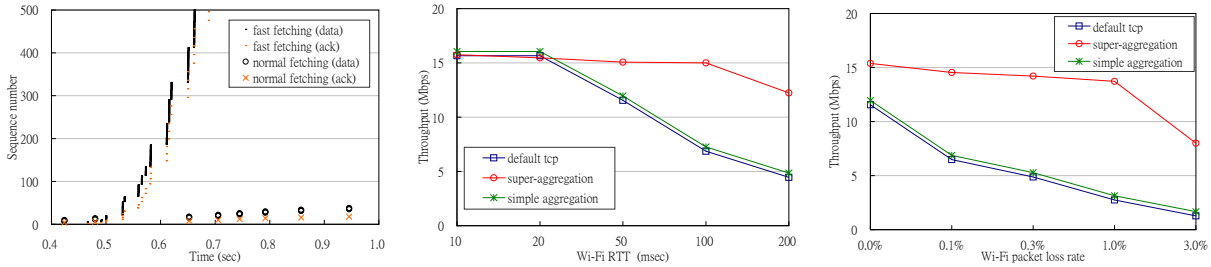
3.7.5 Mirroring-loss-fetching Performance

Figure 9(a) shows that Fast Fetching technique is effective in fetching lost packets using the EVDO interface. It can recover lost segments 36 times faster than naive



(a) Effect of Wi-Fi data rate (b) Effect of capacity in secondary interface (c) Effect of blackout duration

Figure 8: Comparison of *proxying-blackout-freeze* analysis and experiments

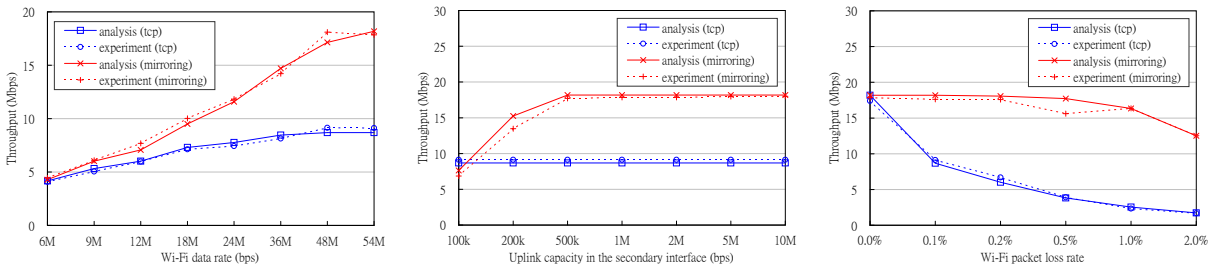


(a) Fast fetching (b) Effect of packet loss rate (c) Effect of RTT on Wi-Fi path

Figure 9: Performance of *mirroring-loss-fetching*

fetching with a normal TCP connection. As shown in Figure 9(b), *mirroring-loss-fetching* significantly outperforms default TCP. It keeps TCP throughput close to loss-less environment until loss rate is more than 3%. Figure 9(c) shows that *mirroring-loss-fetching* gives more improvement magnitude when the Wi-Fi path has longer RTT since TCP spends more time in recovering its congestion window. It improves TCP throughput by 175% when Wi-Fi RTT is 200 msec.

Figure 10 compares the analysis and experimental results of both default TCP and *mirroring-loss-fetching* under random wireless loss. The default packet loss rate is 0.1%. The matching of the analysis and the experimental results shows that the analytical model is very accurate in estimating the throughput under random wireless loss.



(a) Effect of Wi-Fi data rate (b) Effect of capacity in secondary interface (c) Effect of packet loss rate

Figure 10: Comparison of *mirroring-loss-fetching* analysis and experiments

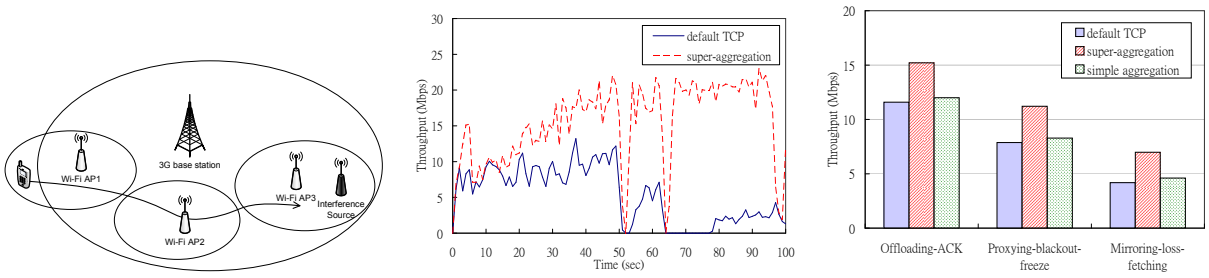
3.7.6 Performance of Integrated Operations

Figure 11(a) shows the experimental scenario for integrated operations. At first only Wi-Fi is available, then the client enters coverage of 3G network. It experiences two-second blackout when it moves out from first Wi-Fi network, across second one, and enters third Wi-Fi network. There is an interference sources in the third Wi-Fi network that causes one-percent random wireless losses.

Figure 11(b) shows instantaneous throughput of *super-aggregation* and default TCP as the client moves along. They have similar throughput when Wi-Fi is the only access at first, and *super-aggregation* starts to outperform default TCP once 3G is available because of *offloading-ACK*. Both throughputs drop to zero during blackouts, but *proxying-blackout-freeze* can immediately resume throughput while default TCP falls back to slow start. *Mirroring-loss-fetching* provides most significant improvement when random wireless losses happen in AP3. Overall, *super-aggregation* gives 189% improvement, which almost triples default TCP's throughput.

3.7.7 Performance on Google Android

Figure 11(c) shows throughput improvements of all super-aggregation principles on Google Android phone with 802.11g: 31% with *offloading-ACK*, 42% with *proxying-blackout-freeze* under 2-second blackouts, and 67% with *mirroring-loss-fetching* under 0.3% packet loss rate. Although throughput achieved on Android is typically lower



(a) Experimental scenario (b) Performance of integrated operations (c) *Super-aggregation* on Android phone

Figure 11: *Super-aggregation* integrated operations and performance on Android phone

than that of laptop, which may be due to different hardware capabilities, improvements from *super-aggregation* principles are significant.

3.8 Issues

IP spoofing: *Offloading-ACK* and *proxying-blackout-freeze* require IP spoofing, which might be blocked by ingress filtering at routers. Experiments in [38] shows that around 25% of ASes allow spoofing, and 40% of clients in the Internet can spoof addresses up to a /8 netblock. There are two ways to support *offloading-ACK* and *proxying-blackout-freeze* when spoofed packets are blocked by ingress filtering. One is to tunnel packets to a proxy that can do IP spoofing. The proxy model is practical since forwarding light traffic for *offloading-ACK* and *proxying-blackout-freeze* can provide significant improvement. The other way is to introduce a new TCP option to allow sender to recognize other IP addresses belonging to the same wireless client.

Layer separation violation: Scope of *super-aggregation* principles span from link layer to transport layer, so they need to be carefully designed to not violating layer separation. We base the design of all three principles on standard operations and common properties of TCP, Wi-Fi, and 3G technologies. All principles work as enhancement to TCP with knowledge of underlying interfaces. It takes consideration to implement *super-aggregation* principles when functionalities in transport layer and

link layer have non-standard implementation.

Extra resource consumption: The generic *mirroring* principle consumes extra resources when establishing the mirroring connection to server. The design of each *mirroring* principle should request for essential contents on the mirroring connection if possible. For example, *mirroring-loss-fetching* uses byte-range fetching technique if the functionality is supported by application-layer protocol of the traffic. This requires knowledge of application-layer functionalities and protocols, and it should be considered as add-on to principles implementation for better efficiency.

Connection mirroring: *Mirroring-loss-fetching* is mainly designed for state-less content services and should not be used in the following cases. If end-to-end semantics is critical for the application on top of TCP, hiding losses from sender side may cause inconsistency issues between two ends. It also assumes server to send identical data when receiving identical requests, but practical servers may give different response if the content provisioning includes randomness or other external information, such as system time. Besides, the mirroring connection cannot be established if the encryption used in the original connection have a different session key for each new connection. For those cases, *mirroring-loss-fetching* cannot be used and *super-aggregation* doesn't modify ACKs generated by the wireless client.

Extension to other wireless technologies: *Super-aggregation* can be applied to any combination of wireless technologies, as long as the interfaces exhibit heterogeneity in terms of three characteristics: capacity, connectivity, and loss rate. Since the different interfaces are likely to operate on different channels (to leverage the multiple interfaces in the first place) and hence will connect to different APs, they naturally will have uncorrelated connectivity and packet losses. Hence, there arise two possibilities based on whether or not there exists capacity heterogeneity: First, when two interfaces have heterogeneous capacities, the rule of thumb that must be applied is that the one with higher capacity acts as the primary interface. The other

interface acts as the secondary to enhance performance through the *super-aggregation* principles. We do note that impact of the degree of heterogeneity on the performance gains with respect to simple aggregation is an interesting problem and something we leave for future research. Second, if the two interfaces have similar capacities, the interface with better connectivity is picked as the primary interface. The other interface is used as the secondary for the *super-aggregation* principles. Note that there may remain unutilized capacity on the secondary interface, and a simple aggregation technique can then be applied to use up the remaining bandwidth.

Thus, considering typical examples of recent wireless technologies such as 802.11n, WiMAX, 3G, and Bluetooth, we believe that the proposed principles will apply as-is to any combination of two heterogeneous wireless interfaces. For homogenous interfaces the applicability again is valid as long as capacity heterogeneity exists: for example, when two Wi-Fi interfaces on the same mobile device use different channels, and have different signal quality and hence data rates. If capacity is homogenous, such as two Wi-Fi interfaces with the same data rate, a combination of *super-aggregation* and simple aggregation will be required to maximize the throughput.

Extension to three or more interfaces: Thus far in this chapter, we have focused on the *super-aggregation* principles applying to only two interfaces. We now briefly discuss how the principles may be extended to apply to three or more interfaces: First, each mechanism is assigned to a different interface based on individual characteristics. Second, if an interface is underutilized, it is used to share the load of another interface. Finally, if any interface is still underutilized, simple aggregation is used along with *super-aggregation* to maximize throughput. This can be exemplified with a mobile device equipped with four interfaces using 802.11n, WiMAX, 3G and Bluetooth technologies respectively. For such a scenario, 802.11n will be selected as the primary interface for the highest capacity supported. WiMAX will then be assigned to *mirroring-lost-fetching* because of its relatively higher capacity. Bluetooth

will be assigned to *proxying-blackout-freeze* for its short latency and low bandwidth, and 3G will be assigned to *offloading-ACK*. WiMAX may have extra uplink capacity since it is mainly used for downloading. It will then be assigned to share the loads of *offloading-ACK* on 3G. TCP ACKs with spoofed IP will be sent via both WiMAX and 3G, according to their uplink capacity. If WiMAX still has unutilized capacity, some data traffic will be split to it by using simple aggregation.

Battery lifetime: Although super-aggregation principles activate multiple interfaces simultaneously, we believe they don't consume more power than when using a single interface. Since throughput is improved by *super-aggregation*, a given amount of data will be transferred faster and energy saved with more sleep time. In this context, we have studied the energy consumed by downloading 10 MB of data on the Android phone with the following energy model, where the parameters are defined as follows: E_m^x is energy consumption per minute to maintain a connection on interface x (with or without power saving); E_t^x is energy consumption per byte transferred on interface x; T is end-to-end throughput; N_p and N_s are number of bytes transferred on the primary interface and that on the secondary one, respectively.

$$E = (E_m^{wifi} - E_m^{wifi-psm}) \cdot \frac{N_p}{T} + E_t^{wifi} \cdot N_p + E_t^{3g} \cdot N_s \tag{37}$$

Based on the empirical measurements of energy consumption in [70] (HTC Wizard in Table 2), *super-aggregation* (with the *offloading-ACK* mechanism) consumes 65.69 joules, which is better than that of default TCP (65.82) and simple aggregation (79.51).

CHAPTER IV

A REMOTE COMPUTING PROTOCOL FOR HETEROGENEOUS DEVICES

4.1 Overview

In the previous chapter, we have presented *super-aggregation* as a rapid application delivery protocol that effectively leverages heterogeneous wireless interfaces available in smartphones. In this chapter, we consider the problem of remote computing protocol for heterogeneous devices, which is a promising approach in rapid application mobilization. We propose a remote computing protocol called *MORPH*, **Mo**bile **R**emote computing **P**rotocol for **H**eterogeneous devices. Traditional remote computing protocols assume homogeneous devices to be used at both ends of the remote session and thus have poor performance when being used from smartphones. *MORPH* addresses the device heterogeneity problem by transforming application views to ones that are more suitable for smartphones. We present the design of a core component of *MORPH* called virtual view that virtualizes application views into an abstract representation irrespective of the UI framework used in the PC applications. Then, we introduce transformation services that can be applied onto the virtual view to realize smartphone-friendly views. In the next two chapters, we will elaborate on three core transformation services, which are aggregation, translation, and traffic suppression.

4.2 Remote Computing for Heterogeneous Devices

Since *MORPH* is built on top of traditional remote computing, we present a short primer on the technology and its applications in mobile computing. Figure 12 illustrates the network model of remote computing for heterogeneous devices where the

server resides in a PC and the client resides in a mobile device, such as a smartphone or a tablet. Virtual network computing (VNC) and Remote desktop protocol (RDP) are both examples of remote computing protocols. In VNC the server sends raw pixel information, while in RDP the server sends graphical primitives and commands. Remote computing has several application scenarios. First, using remote computing, a mobile user can access his or her own PC when being away from it. It would be very useful when the user wants to access certain applications or data that exist in the remote PC but not the smartphone. A recent study [57] shows that knowledge workers spend only 35% of work time at their desk. When the users are away from their PCs for the remaining 65% of the time, being able to access the PCs from smartphones would provide great convenience and boost their productivity. Second, remote computing allows a user to access a virtualized PC. For example, VDI is a popular technology used in industry to virtualize the desktop environment for enterprise employees. Being able to access a VDI system from a smartphone again provides convenience and productivity benefits. Third, remote computing allows a user to access others' PC such as in remote IT support that an IT technician can remotely access a user's PC from a smartphone to provide troubleshooting in real time.

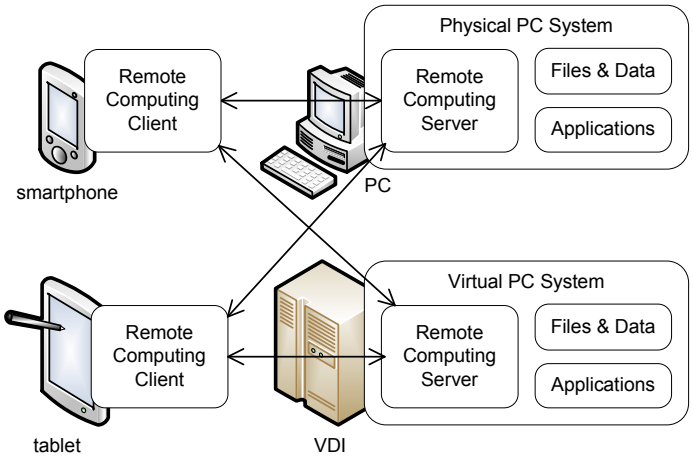


Figure 12: Remote computing from heterogeneous devices

4.2.1 Remote Computing for Application Mobilization

Remote computing can be used as a mobilization strategy by simply running the target application on a PC backend, and providing a view of that application's bounding box to the smartphone. Attaching the backend PC to the appropriate file system can provide access to the user's data stores. We use VNC as the underlying remote computing protocol of the baseline solution due to the ready availability of open-source VNC clients. However, the design presented is agnostic to the specific remote computing protocol. There are three advantages to using remote computing for mobilization:

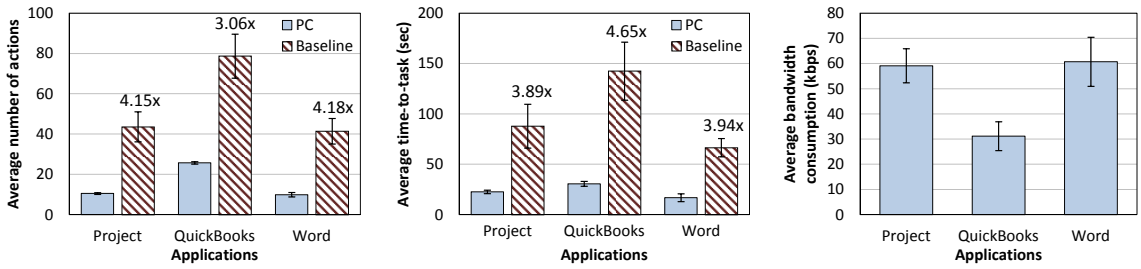
- **Zero software porting:** Perhaps the most important advantage of mobilization using remote computing is that there is no porting of the application software required. The application still runs only on a PC, and merely a view is furnished to the smartphone.

- **Easy IT manageability:** A by-product benefit of continuing to use PC applications even for smartphone users is that the management of the mobilization infrastructure can be done using existing IT processes, such as those for software updates.

- **Familiar interface:** Since users have previously relied on the original PC application, the interface furnished through remote computing will continue to be familiar, and any functionality accomplishable using the PC could be performed from the smartphone.

4.2.2 Challenges with Remote Computing

The biggest drawback with using remote computing for mobilization is that the *application view from the PC is presented as-is on the smartphone without any transformations save for resolution scaling.* The PC view is quite cumbersome to use on the smartphone, and several reasons contribute to the unwieldiness of the interface:



(a) Task effort (b) Time-to-Task (c) Traffic Consumption

Figure 13: Performance of Baseline Remote Computing

(i) The bounding box of the application on the PC is typically much larger than the screen real estate on the smartphone. This raises a pan/zoom trade-off for the user. In a zoomed out mode, panning to reach different sections of the view is reduced, but the UI elements are too small to read or manipulate easily. Thus, once the user reaches the section of interest, a zooming in is almost always required. In a zoomed in mode, the user has better visibility of the UI elements, but the burden to pan increases considerably. (ii) Application interfaces on smartphones are typically layered for better navigation and organization. PC application interfaces typically have a flatter structure with fewer layers but a denser element layout. Such interfaces are not suitable for smartphones and may cause extra burden on mobile users. (iii) Independent of the above issues that increase user effort, performing the same number of actions on the smartphone as on the PC is also subjectively burdensome to the user due to the constrained environment.

To study the above problems quantitatively, we perform user-studies of three applications - Microsoft Project, Intuit Quickbooks, and Microsoft Word - mobilized using baseline remote computing. We measure the objective metric of total number of *actions* (such as mouse clicks/keyboard entries) taken by each of the 10 participating users to complete pre-defined tasks. Figure 13(a) shows the performance when the tasks are performed on the smartphone and PC respectively (with 90% confidence intervals). The average number of actions required on the smartphone is $3.1x$ to $4.2x$

of that required for the PC. Similarly Figure 13(b) shows the average time required to perform the same tasks. Such inflation in user burden can directly be attributed to the reasons identified earlier.

Another drawback of remote computing is the back and forth exchange of data between the server and client that could impose data usage burdens on the wireless link. Hence, we analyze the traffic consumed by the baseline remote computing protocol for the three PC applications averaged over the six tasks per application. The average traffic consumption for the three applications is presented in Figure 13(c) and is 50.33 Kbps during the execution of a task. While not prohibitive, the performance would be an issue in cellular data networks where the available per user capacity can dip below the required data rates. Furthermore, with increasing trends toward usage based billing in cellular data networks, reducing the required data rate will also ease cost burdens.

4.2.3 Problem Statement

In the rest of this chapter we answer the following question: *Could transformations be applied to remote computing views to make smartphone users more effective when accessing a remote PC?* In answering this question, we tackle the challenges identified for remote computing earlier in the section.

4.3 View Virtualization

MORPH is a remote computing protocol for heterogeneous devices that enables application view transformation on top of traditional remote computing. Thus, it inherits the various advantages of remote computing discussed in Section 4.2. More importantly, it is designed to explicitly address the challenges identified. At a high level *MORPH transforms the application view dynamically for the smartphone environment* within the context of the remote computing session. The transformation is performed in an *application-agnostic fashion* with the goals of *reducing user-burden and traffic*

consumption.

The design of *MORPH* is predicated on two properties of typical applications: (i) Application user-interfaces (UIs) are built by relying on well-known frameworks for the creation of UI elements. Examples of such UI frameworks include Microsoft .NET, SAP GUI, Java, Web DOM, GTK+, Cocoa, QT, and Flash. (ii) Accessibility frameworks exist for each of the above UI frameworks that allow for UI elements to be monitored and manipulated.

At a high level, the *MORPH* design (see Figure 14) consists of two components: the *view virtualization* that converts any application view into a virtualized view of UI elements with well defined attributes and APIs; and the *transformation services* that transform the virtual view for the specific smartphone environment. In the rest of the section we elaborate on the view virtualization and describe several transformation services in Section 4.4.

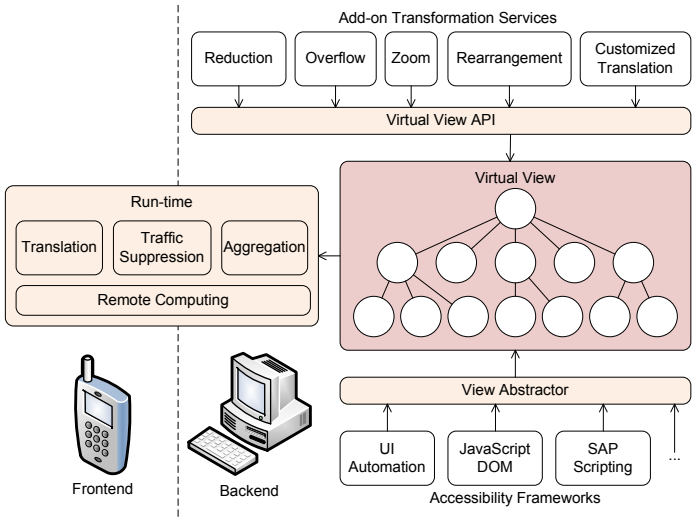


Figure 14: *MORPH* system overview

The goals of creating a *virtual view* are two-fold: (i) it shields the complexity of interfacing with the myriad of UI platforms from the transformation services, and instead exposes a standardized representation of the UI elements that the transformation services can manipulate; and (ii) once a *virtual view* is created, different sets

Table 3: Virtual View API and mapping to accessibility frameworks

UI framework		Microsoft .NET	SAP GUI	Web DOM
Accessibility framework		UI Automation	SAP GUI Scripting	JavaScript and DOM
Attributes				
	Id	{Name, LocalizedControlType, AutomationId}	{Name, Type, Id}	{ Name, TagName, Id}
	Type	LocalizedControlType	Type	TagName
	Location	{Left, Top}	{Left, Top}	{OffsetLeft, OffsetTop, Offset-Parent}
	Size	{Right-Left, Bottom-Top}	{Width, Height}	{OffsetWidth, OffsetHeight}
	State	Value and State	Value and State	Attributes
	Parent	Ancestors.head	Parent	ParentNode
	Children	Children	Children	ChildrenNode
	Status	N/A: virtual view-only		
	Template	N/A: virtual view-only		
Functions				
	Read()	Read from an attribute		
	Write()	Write to an attribute		
	Invoke()	Invoke	Execute	Click
Events				
	OnOpen()	MenuOpenedEvent, WindowOpenedEvent	ContextMenuEvent, ChangeEvent	DOMNodeInserted, DOMNodeInsertedIntoDocument
	OnClose()	MenuClosedEvent, WindowClosedEvent	DestroyEvent	DOMNodeRemoved, DOMNodeRemovedFromDocument
	OnActivity()	FromPoint, FocusedElement	FindByPosition, GuiFocus	OnClick, OnDblClick, OnKeyUp

of transformations can be applied to the same view in a smartphone platform specific manner. Thus, the *virtual view* has to be simple but flexible enough to facilitate the creation of powerful transformation services with relative ease.

The *virtual view* in *MORPH* is an abstract representation of the UI of a PC application. It is represented as a tree where each node corresponds to a UI element in the application, and each link represents the relationship between a container UI element and another UI element in the container. Specifically, the root node of a *virtual view* represents the main window of a PC application. The window typically contains a menu bar, a tool bar, a status bar, and other UI elements that are all represented as children to the root node. The children in turn contain other children. For example, the menu bar contains several menu items, which may further contain sub-menu items.

For each node in a *virtual view*, a set of attributes is defined that describes the UI element. Table 3 lists nine attributes that are extracted from the application view by

the UI abstractor - (i) *ID* is a unique identifier for the UI element; (ii) *type* represents the specific type of a UI element such as button, menu item, text field, etc. (iii) *location* describes the coordinates of the UI element in the application view; (iv) *size* contains the width and height of the UI element; (v) *state* represents information contained inside the UI element, such as the text in a text field or the checked state of a check box; (vi) *parent* and *children* describe the hierarchy relationship in the tree structure of the virtual view; and finally (vii) *status* indicates the visibility of the UI elements while *template* provides style descriptions when rendering the UI element.

A *virtual view* also contains dynamic information in terms of the events triggered in the PC application. As shown in Table 3, three types of common events are captured: - (i) an *OnOpen()* event occurs when a view, such as a pop-up menu or a dialog, is opened in the PC application; (ii) an *OnClose()* event occurs when a view is closed in the PC application; and (iii) an *OnActivity()* event occurs whenever the user performs activity, such as a mouse click or a keystroke, on a UI element. Finally, besides attributes and events, the virtual view provides a simple API to allow manipulation of the UI elements. As shown in Table 3, the API provides three functions: the *Read()* and *Write()* functions allow a service to access the value of an attribute of a UI element, and the *Invoke()* function allows a service to perform activity on a UI element, such as the clicking of a button element.

The *MORPH UI abstractor* is the component that interfaces with the different UI frameworks in extracting attributes for the *virtual view*, monitoring for events to report through the virtual view, and invoking appropriate framework specific APIs to support the virtual view functions. The *UI abstractor* creates a uniform representation of virtual views by leveraging the accessibility frameworks associated with the UI frameworks. Table 3 shows how the attributes and events in virtual view are matched with those used in three popular UI frameworks - UI Automation, SAP GUI scripting, and JavaScript.

4.4 Transformation Services

In the previous section, we have present the view virtualization of *MORPH* that converts the application view of any PC application into a virtual view. In this section, we present transformation services that can be programmed into *MORPH* to convert the virtual view into smartphone-friendly views. Among the transformation services of *MORPH*, we focus on three *core services* that deal with fundamental challenges that have been discussed in Section 4.2.2. We then present *add-on services* that are optionally added depending on requirements.

4.4.1 Core Transformation Services

The three core services are *translation*, *aggregation*, and *traffic suppression*. The first two services deal with the usability challenges in accessing a PC application via remote computing where the application view is not originally designed to be consumed from a smartphone. The last service deals with the traffic consumption of *MORPH* that relies on remote computing to deliver the application view.

Translation: This service transforms the *virtual view* into an application view that is appropriately for the smartphone. It translates each UI element in the *virtual view* into a smartphone native UI element if a corresponding widget module is available in the smartphone platform. For example, mouse clickable tabs could be transformed into touchable buttons, and a pull down menu could be transformed into a spinner wheel. The action performed on the native UI element is translated back into a corresponding event in the PC application. If no native UI element in the smartphone platform matches with an element in the virtual view, the element is categorized as a workspace item and is rendered as-is graphically through the underlying remote computing. The action performed in the workspace is also redirected to the PC application via remote computing.

Aggregation: This service allows a user to aggregate multiple actions in a task into a single click invocation when using a mobile app transformed by *MORPH*. For routine tasks that the user wants to execute in a mobile app, aggregation service allows the user to record them in to a single macro. A macro can be then accessed from the *MORPH* frontend to automate and also speed up the corresponding task execution. The service can also be extended further to support parameters or custom values for certain actions during playback of a macro.

Traffic Suppression: This service efficiently delivers the *virtual view* of an application transformed by *MORPH* to the frontend client. As described in the translation service, a virtual view may contain elements that requires graphical rendering. Thus, the traffic suppression service is built on top of an existing remote computing protocol, such as RDP or VNC, to enable delivery of *virtual view*. However, existing remote computing protocols are not explicitly designed for mobile applications transformed from their PC counterparts, so there is unnecessary traffic consumption involved in the *virtual view* delivery process. The traffic suppression service is explicitly designed to leverage information available in the *virtual view* to suppress unnecessary traffic in the underlying remote computing without compromising its functionality.

4.4.2 Add-on Transformation Services

In this section, we introduces several feature-rich *add-on transformation services* can be built into the mobile app transformed by *MORPH*. Figure 15 illustrates five example services that we use to demonstrate the flexibility of the *virtual view*.

Reduction: This service provides a simplified view of the mobile app by showing only a subset of the UI elements that the user wants to access from the smartphone. The reduction can be performed by one of the following ways: a) manually choosing the useful UI elements or b) automatically choosing only the frequently used subset of UI elements. For manual reduction, the service can provide an "edit mode" by

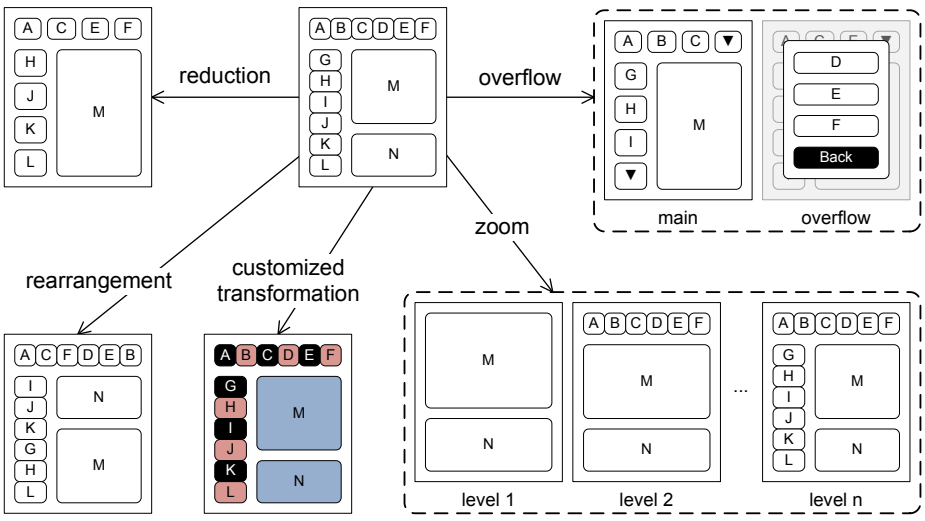


Figure 15: Concept of add-on services

adding a "show/hide" toggle button adjacent to each UI element.

Overflow: This service intelligently splits a virtual view into multiple views to suit the smartphone environment. Because of the difference in form factor, the user interface of a PC application typically has more elements than that of a mobile app. When a virtual view contains more elements than what can be fit into a smartphone screen, squeezing all elements into the small screen would result in a dense and unusable user interface. Overflow service intelligently organizes elements in a virtual view, presents a suitable number of elements for the smartphone screen, and hides the remaining elements in an "overflow" view visible on demand.

Zoom: This service allows a user to dynamically adjust the set of UI elements available in the mobile app from the frontend on demand. It allows the user to define different zoom levels for a virtual view, ranging from simple but feature-limited ones with fewer elements, to feature-rich but complex ones with more elements. It provides a control knob in the frontend to allow the user to dynamically adjust the zoom level in run-time depending on what features the user wants to access.

Rearrangement: This service presents a customized layout for the transformed view and allows configuration of the placement of the various UI elements in the view.

This is particularly useful if the usage patterns of the mobile app are different from those on the PC application and thus the original layout is no longer suitable.

Customized Translation: The service allows users to further customize the look-and-feel of a mobile app transformed by *MORPH*. Specifically, it allows a user to customize how a UI element in virtual view is translated into a native element in the smartphone platform.

CHAPTER V

AN EFFECTIVE REMOTE COMPUTING SOLUTION FOR SMARTPHONES

5.1 Overview

In the previous chapter, we have presented the view virtualization of the *MORPH* protocol and have introduced transformation services that can be built on the *virtual view*. In this chapter, we present the design of the aggregation transformation service that reduces the time and effort required in accomplishing tasks from smartphones. Aggregation allows a user to combine multiple actions into a single click invocation to reduce the effort and time in executing a routine task. We introduce a key building block called *smart macros* that have the robustness of application macros but at the same time possess the generality of raw macros. Using smart macros, we design and prototype $MORPH_{Aggregation}$, a remote computing solution for smartphones. We show using experimental studies and a trace based analysis of real user activity that $\text{MORPH}_{\text{Aggregation}}$ can improve user experience considerably.

5.2 Motivation

While VNC and other remote computing solutions are mature and effective solutions to provide remote access to a PC from *another PC*, they are not explicitly designed for remote access from *a smartphone*. Remote access from a PC is very intuitive to an user, since the local PC provides a homogeneous user interface as the remote PC. The full screen display is shown on the local monitor, and the user controls the remote PC using a mouse and a keyboard at the local PC. In such a scenario, the overall user experience of remote computing is close to that of using a local PC. However,

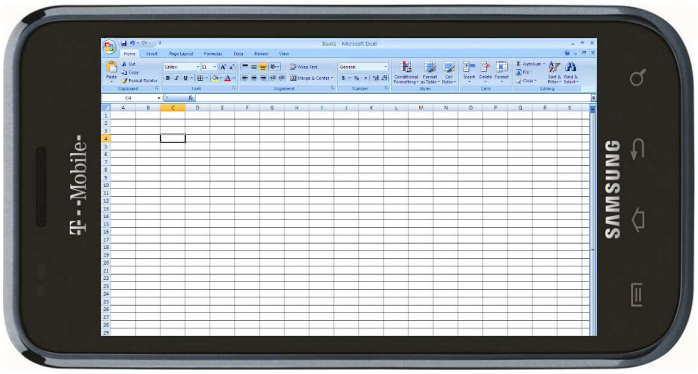


Figure 16: Screenshot of a smartphone VNC client shows an intricate interface.

remote computing from a smartphone is significantly more difficult because of the constraints in the device including form-factor, screen size, and the lack of a mouse and a keyboard. We characterize the degree of cumbersomeness with the *task effort*, which is defined as the number of operators needed to accomplish a computing task. It can be represented with the following equation:

$$TaskEffort^{VNC} = TaskEffort^{PC} \times Inflation \tag{38}$$

where *Inflation* is the factor representing the additional burden imposed on the user by the limitations of the smartphone. In the rest of this section, we first provide qualitative reasons causing such *Inflation*, and then we analyze PC usage traces collected from different users and show that there are considerable amounts of usage redundancy to be leveraged to reduce $TaskEffort^{PC}$. In Sections 6.2 and 5.4, we show how both factors are addressed by $\text{MORPH}_{\text{Aggregation}}$.

5.2.1 Inflated Effort in Remote Computing from Smartphones

We explain how the task effort is increased in remote computing from a smartphone with the following (non-exhaustive) reasons: (i) *The zoom problem:* By default, the full desktop screen is squeezed in to the small screen on the smartphone, as Figure 16 shows a screenshot of MS Exel via AndroidVNC [1]. This renders it almost impossible to directly access or manipulate any single GUI element on the desktop. Doing so

requires the user to zoom in, which is commonly done using *pinching* on newer smartphones. However, such zooming requires the user to perform additional operators. (ii) *The pan problem:* Zooming comes with a by-product problem. Once the screen is zoomed in, not all GUI elements on the original desktop screen is now visible to the user, and panning is required to navigate the full screen. This typically involves the user swipe across the screen and further increases the number of operators the user has to perform. (iii) *The keyboard problem:* Many smartphones do not support a full fledged keyboard. Hence, to access keys not available in the default layout, users will have to press additional buttons on the keyboard. For example, on the stock Android keyboard, users will have to press the '123' button to access the number keys. Some keys are specific to PC and are unavailable in smartphone keyboard at all, such as 'Ctrl' and 'F1' that requires a custom implementation in the remote computing client to perform. (iv) *The error problem:* Users tend to make more mistakes on the smartphone when performing operators. Undoing such mistakes and re-performing the operators would require extra operators. $\text{MORPH}_{\text{Aggregation}}$ is explicitly designed to reduce *Inflation* but we defer discussions on how it accomplishes the reduction till later in the chapter.

5.2.2 Measurement of Redundancy in User Activity

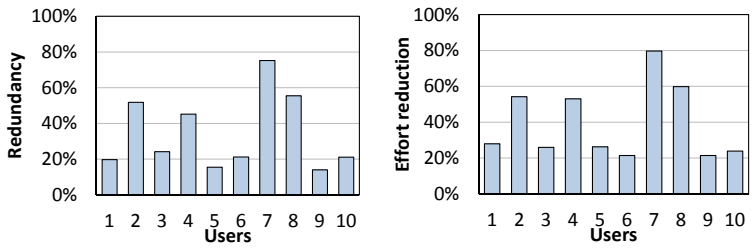
While reducing *Inflation* is one approach to reduce $TaskEffort^{VNC}$, another complimentary approach we consider is the reduction of $TaskEffort^{PC}$ by enabling to perform operators in aggregates. Such *operator aggregation* would specifically be relevant if users naturally tend to perform *redundant aggregates of operators*. Before we delve any further into ways to accomplish this, we now briefly present results from a user activity analysis to study whether there is redundancy in user operators.

The PC user activity trace is collected from ten different volunteer users spanning both academia and the industry. Each user is provided with a custom-built monitor

utility that captures all operators performed by the user and exports them onto a stored file. The monitor transparently runs in the background on the PC so that it captures the true activity of the users in their routine usage of the applications. Each user periodically emailed back the stored file and the usage analysis was performed offline.

The usage analysis first involved analyzing the redundancy in user activity, or in other words the amount of repetitive activity that can be reduced by operator aggregation. We define the degree of redundancy as follows: Consider an operator sequence "ABCABCDABE". This contains two repetitive substrings: "ABC" and "AB". Each repetitive substring can be replaced with a new *code*, and the resultant string will reveal the upperbound of the redundancy elimination possible. For example, "ABC" can be replaced with the code X and "AB" can be replaced with the code Y. The original string can thus be reduced to "$XXDY$E", which reduces the length from 10 to 5. The redundancy in the sequence is then calculated to be $\frac{10-5}{10} = 50\%$.

To quantify the redundancy in user activity, we need to first discover repetitive tasks. We define a repetitive task to be a sequence of operators that has appeared in the history at least twice. To analyze the redundancy, we use the user history to perform trace-based evaluation. We greedily match the history with the repetitive tasks that have been identified, and we assume that each repetitive tasks can be replaced with a single new operator. We consider only repetitive tasks of at least length two for such replacement. Figure 17(a) shows the redundancy of different users participated in the analysis. The redundancy in user activity ranges from 20% to 40% for most users, and the average is 30.96%. While showing an exhaustive list of repetitive tasks we identified in the traces is prohibitive, we provide a few example repetitive tasks: 1) In MS Excel printing a chart in to a colorful ps file; 2) In MS Word changing the spacing between paragraphs with a specific pt. *This indicates that an intelligent operator aggregation technique can reduce task effort in*



(a) Redundancy observed in user activity

(b) Potential effort reduction on a smartphone by user

Figure 17: Real-user activity shows redundancy and potential effort reduction

a significant fashion.

There are two relatively offsetting factors that have to be considered when translating the above degree of redundancy into what degree of effort reduction can be achieved on the smartphone: (i) The above analysis assumes that replacement code is always of length one, which obviously is not realistic. Hence, a notion of *lookup overhead* for the newly introduced codes has to be incorporated. We consider the lookup overhead to be approximated as $log_k N$ where k is the number of aggregate operators that can be presented to the user at any given point in time, and N is the total number of such aggregate operators. (ii) The analysis also does not account for the mobile inflation factor discussed earlier. Using the remote computing experiments conducted later in Section 6.4 where users were asked to perform the same tasks on both the PC and the smartphone, we determine the mobile inflation factor to be 3.31. Figure 17(b) shows the effort reduction after the adjustment of both the lookup overhead and the mobile inflation factor. The potential effort reduction on the smartphone ranges from 20% to 80%. The effort reduction is related to the redundancy in user activity, which is intuitive. The interesting observation is that the effort reduction can be higher than the redundancy that appears in the user activity in PC. The reason is because a set of aggregated operator can be presented as a simple button and avoid the operator inflation effect. Analysis across top applications in the trace shows a similar reduction potential from 20% to 80%.

5.3 Design Principles

5.3.1 Overview

$\text{MORPH}_{\text{Aggregation}}$ is a software solution designed to improve the experience of a user accessing a remote PC from a smartphone. It is a two-ended solution, with presence at both the PC and the smartphone (see Figure 18). The $\text{MORPH}_{\text{Aggregation}}$ server at the PC co-exists transparently with a remote computing server. The $\text{MORPH}_{\text{Aggregation}}$ client at the smartphone is integrated as an overlay with the remote computing client. $\text{MORPH}_{\text{Aggregation}}$ provides a powerful framework for *users to create robust, general and extensible macros* on the PC, *name them*, and *invoke them easily at the smartphone within the context of the remote computing client*. In the rest of this section we focus on the fundamental design elements in $\text{MORPH}_{\text{Aggregation}}$: *application agnostic smart macros* that allow $\text{MORPH}_{\text{Aggregation}}$ to provide the robustness of application macros but with the generality of raw macros; *task effort reducing front-end* on the smartphone that is presented as an overlay within the context of the remote computing client; *parameterization and pre-emptability* of macros that provide a high degree of flexibility and extensibility; and *offline macro recommender* that analyzes user activity and provides recommendations for macros to be created. Note that the rest of the discussion in this section is not meant to be an exhaustive description of $\text{MORPH}_{\text{Aggregation}}$, but rather the principal design elements. We defer the complete description to the next section.

5.3.2 Application-Agnostic *Smart*Macros

Operator aggregation through the creation of *macros* is a desirable capability for task effort reduction. However, as explained in Section 5.2, traditional approaches to creating macros have been severely limited by one of two problems: raw macros suffer from robustness issues and are notoriously erroneous if the playback environment differs even slightly from the recording environment; and application macros suffer

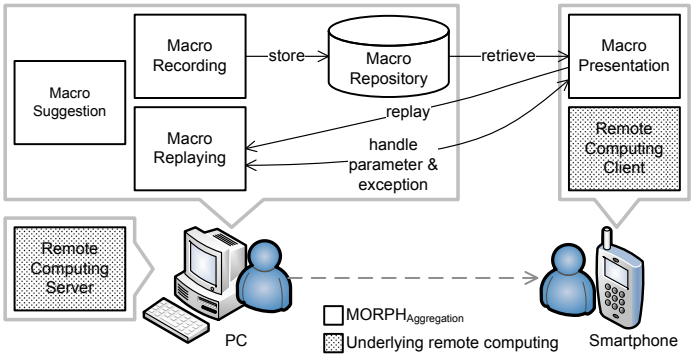


Figure 18: $\text{MORPH}_{\text{Aggregation}}$ system overview

from a lack of generality and users are dependent on application developers to provide the capability of macros on a per-application basis.

$\text{MORPH}_{\text{Aggregation}}$ uses a new type of macros that we introduce called *smart macros* that have the robustness of application macros but provide the generality of raw macros. The design of smart macros is derived from a combination of key trends and observations on how operating systems today work. Most operating systems (OS) today provide application developers with higher order primitives called *GUI elements* than simply the ability to create graphical objects. Thus, application developers can readily use GUI elements such as text-boxes, buttons, and forms instead of creating them from scratch. Because the OS also assumes the responsibility of providing callbacks to the application when GUI elements are invoked or manipulated, applications register each GUI element with well identifiable information ranging from the parent application to the specifically recognizable name of the element to the state of the element. *Smart macros tap into the GUI element framework directly to facilitate the recording of robust macros in an application agnostic fashion.* Thus, a smart macro at a high level is a sequence of operators with each operator represented as an addressable GUI element with its appropriate state and the raw user input to be delivered to that GUI element.

Figure 19 shows how smart macros compare to the other types of macros. As

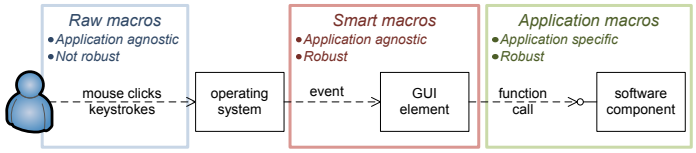


Figure 19: Types of macro solutions

we explain in Section 5.4, the UI automation and the .NET frameworks available in MS Windows are used in tandem for tapping intelligently into the GUI element framework and capturing raw user activity. We also discuss how such frameworks are also readily available on other OSes.

5.3.3 Task Effort Reducing Front-end

Once *smart macros* are created, they have to be presented to the user on the smartphone. $\text{MORPH}_{\text{Aggregation}}$ uses a push technique to update the list of macros available on the smartphone, but we defer further discussion of the update mechanism to the next section. More importantly, the actual front-end on the smartphone has to be designed carefully with the following considerations: (i) the front-end has to be *non-intrusive* and should ideally seamlessly co-exist with the remote computing client front-end; (ii) the front-end must be *non-limiting* in terms of what the user can accomplish independent of whether relevant macros are available or not; and (iii) the front-end should be heavily tailored toward reducing task effort.

The $\text{MORPH}_{\text{Aggregation}}$ front-end is designed to address the above considerations. It is designed as a collapsible transparent overlay to the remote computing client on the smartphone (see Figure 23). The user continues to be able to view and use the regular remote computing client and *opportunistically can invoke macros from the overlay macros panel*. The user can collapse the macros panel if required. Creation of the front-end in this fashion is straightforward as the $\text{MORPH}_{\text{Aggregation}}$ client is integrated with the remote computing client software. When a macro is invoked, the invocation is carried over to the $\text{MORPH}_{\text{Aggregation}}$ server on the PC out-of-band of the

actual remote computing session. However, the playback of the macro at the PC is presented back to the user real-time naturally through the remote computing session. Thus, the user can actually observe the execution of the macro. *More importantly, the user can seamlessly intersperse the use of macros with raw input to the remote computing client interface indefinitely.*

Finally, the front-end is also designed to minimize the task effort when invoking macros. When the replay of macros requires user-input (see discussion on Parameterization later in the section), the front-end *automatically pans and zooms* to the concerned GUI element that needs manipulation by the user. This eliminates the need for the user to navigate to that element. Note that because the $\text{MORPH}_{\text{Aggregation}}$ server has the exact information about the GUI element on the PC, identifying its coordinates on the smartphone through the API of the remote computing client is straightforward. Finally, if the GUI element can take default values (e.g. text-boxes, check boxes, radio buttons, etc.) $\text{MORPH}_{\text{Aggregation}}$ pre-populates the GUI element with the default values derived from the state of the GUI element when the macro was recorded or last replayed whichever was later. The front-end provides users with the option of replaying an entire macro with default values in which case the user is not prompted for input during the replay. Finally, the replay of the macro itself is done faster than real-time minimizing the task-time for users.

5.3.4 Parameterization and Preemptability

Operator aggregation using macros is useful when the user wants to replay the exact sequence of operators represented by the macros. However, in reality users could want minor variations in tasks every time a macro is replayed. $\text{MORPH}_{\text{Aggregation}}$ accommodates such variations by supporting *parameterization of the smart macros*. At a high level when the user records a macro the operators of the user can be classified as parameters. The macro is then recorded as a sequence of only the operators with

appropriate indicators for when user input should be sought for the parameters. The original input for the parameters is also preserved as default input values for the macro. This provides a powerful abstraction to users as macros can now be used with variations on the fly. To the best of our knowledge, this is the first approach to enable parameterization of generalized macros.

The classification of an operator into either a parameter or not is done by distinguishing *whether the operator merely changes the state of the associated GUI element or invokes the GUI element respectively.* During playback of a parameterized macro, $\text{MORPH}_{\text{Aggregation}}$ provides users with the option of executing the macro *in continuous mode* with just the default values and not pausing for input. If a user does choose to replay the parameterized version of a macro, an option is still provided to resume the macro in continuous mode after every parameter input.

Another form of macro extensibility $\text{MORPH}_{\text{Aggregation}}$ supports is the ability to pause macros, perform raw input, and then resume the original macro. This provides a further degree of extensibility than parameterization as macros now can be extended with arbitrary introduction of new operators as well. By default, the pausing of macros is allowed in $\text{MORPH}_{\text{Aggregation}}$ only when the playback is stopped for user input. However, it is possible to instrument the playback to occur slower than real-time and allow users to pause the macro at any given point in the playback.

5.3.5 Offline Macro Recommender

The effectiveness of the $\text{MORPH}_{\text{Aggregation}}$ solution depends on the user creating useful macros. While the user is very likely to know the most important aggregates of operators it is challenging for a user to be able to create all useful macros. Hence, one of the important design elements in $\text{MORPH}_{\text{Aggregation}}$ is the offline macro recommender. The $\text{MORPH}_{\text{Aggregation}}$ server monitors user activity even when a macro is not being recorded and logs the activity after classifying parameter operators from

others. The macro recommender component of $\text{MORPH}_{\text{Aggregation}}$ then, on demand, parses the log to generate a list of repetitive tasks observed in the user activity. For each such task, the application, the length, the exact sequence of operators, and the frequency of occurrence of the task are presented. The report is generated simply as readable text. The tasks are first filtered based on user specified length and frequency thresholds, and rank ordered based on decreasing values for $length * frequency$. Users may then choose to explicitly record a subset of the macros specified in the report[1].

The tracking of user activity is done intelligently to account for users switching from one application to another. Thus, if a user performance activity $A1$ at time $t1$ in application $App1$, moves on to another application, and returns back to $App1$ to perform activity $A2$, $\text{MORPH}_{\text{Aggregation}}$ will track the concatenation of $A1$ and $A2$ as a continuous task. Also, the Macro Recommender only finds the longest matched sequences to eliminate redundantly identifying sub-strings of tasks as also repetitive tasks. Briefly, this is accomplished using a suffix tree for building task patterns from the user activity log. The Macro Recommender takes the history of user activity since the last time a recommendation report was generated, and it inserts all suffixes of the history into the suffix tree. After inserting all suffixes, the Macro Recommender traverses the suffix tree to identify repetitive tasks, which are nodes that satisfy both the length (node depth) and the frequency thresholds. We defer the detail mechanism to the next section.

5.4 Solution

In this section we present the system realization of the $\text{MORPH}_{\text{Aggregation}}$ solution. We present the system architecture, the details of our implementation and the various components involved in the building the system.

Figure 20 shows the software architecture and the components that reside in four

[1]Future work could facilitate users simply choosing an identified task and the macro automatically being recorded.

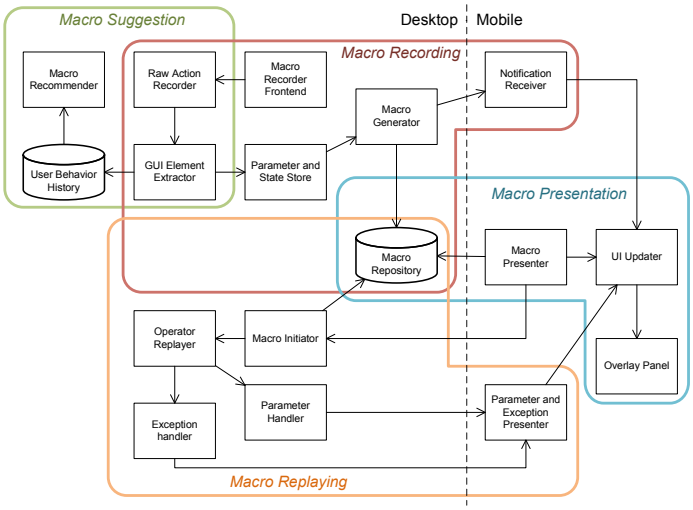


Figure 20: $\text{MORPH}_{\text{Aggregation}}$ software architecture

functional blocks described in Section 6.2. The solution is designed as a server-client architecture, where the server resides on the desktop and the client resides on the smartphone. Since the $\text{MORPH}_{\text{Aggregation}}$ design is centered around operator aggregation that is orthogonal to the core remote computing functionality, the solution is implemented as an overlay to an existing remote computing solution but does not change any of the native behavior of the remote computing software. The $\text{MORPH}_{\text{Aggregation}}$ client requires integration with the remote computing client on the smartphone, but the integration is merely to gain access to the UI and the remote computing protocol and interaction is left untouched. The $\text{MORPH}_{\text{Aggregation}}$ server on the other hand is fully decoupled from the remote computing server and has no direct interactions with it. Also, any communication between the $\text{MORPH}_{\text{Aggregation}}$ client and server is done out of band of the remote computing session. The arrows in the figure show the interaction between components in terms of function calls.

5.4.1 $\text{MORPH}_{\text{Aggregation}}$ Server on Desktop

The $\text{MORPH}_{\text{Aggregation}}$ server resides on the desktop and implements three functional blocks of the solution, namely the Macro Recording, Macro Replaying, and Macro Suggestion. The desktop also maintains a persistent database for all the

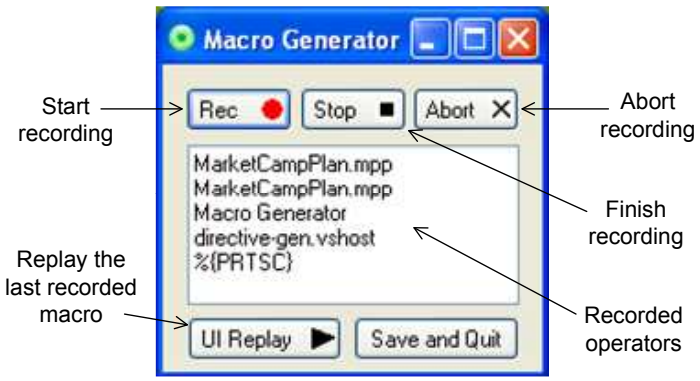


Figure 21: $\text{MORPH}_{\text{Aggregation}}$ desktop UI screenshot

recorded macros. All the desktop components are developed using C#. The persistent database is an SQL-based relational database called HyperSQL [9]. We use an unmodified realVNC [71] server on the desktop as an independent process. While a number of components constitute the $\text{MORPH}_{\text{Aggregation}}$ server we explain the key ones below and only briefly summarize the others.

Macro Recorder Frontend: This is a simple GUI application that allows the user to start/stop/abort the recording of a macro and also allows playback for verification. Figure 21 shows a screenshot of the frontend.

GUI Element Extractor: This component has two responsibilities: converting raw operators into GUI elements and retrieving a unique identity for each element. The GUI element extractor uses the APIs provided by the accessibility frameworks for extracting the handle for the GUI element. In the context of Windows the UIA framework provides functions FromPoint() and FocusedElement() to determine the AutomationElement for a mouse entry and a keyboard entry, respectively. Next, the GUI element extractor has to retrieve a unique identity for the GUI element so that it can be reliably located while executing a macro. The AutomationElement has several properties that could be used to identify it, such as name or automation ID. However, even a combination of these properties is not sufficient to uniquely identify an element. Automation ID is not provided by all GUI elements, and multiple GUI

elements in a GUI application window can have the same name. We propose to trace the GUI tree hierarchy from the target GUI element back to the root and use the full ancestor list as the unique identity.

Parameter and State Identifier: To allow parameterization, we categorize GUI elements as a parameter type based on specific control types (Ex: edit box, check box, radio button and drop-down menu). We record the operator performed during the recording of the macro as the default value for the parameter. We also maintain the state of those GUI element which function is stateful, such as setting/unsetting a check box.

Macro generator: When the user clicks the stop button on the macro recording frontend, all the individual operators are aggregated in to a single macro. The macro is a dynamic array of GUI element information recorded for each operator. Each entry in the array is self-sufficient to replay the necessary operator and the information stored contains the process on which an operator is performed, the unique identity of the GUI element so that it can be retrieved, the GUI element's state and the operator performed. The user can manually provide a name for the macro.

Operator Replayer: The *Operator replayer* is responsible for replaying an individual operation. Since we have already recorded the full path of the GUI element handle in the GUI element tree, we can walk through the tree from the root element of the tree to reach the required GUI element and thus retrieve its handle. In our implementation we use the FindChild() method provided by the UIA library to traverse the GUI element tree. Next, the recorded state of GUI element is restored before the operator is performed on the element. Lastly, the operator is performed on the GUI element. For a mouse click operator, we send a mouse click to to the GUI element retrieved. We use the sendInput() function (available in the user32.dll) to replay a mouse click. Similarly, focus is set to the GUI element that is supposed to receive the keyboard operator and the raw keys are sent to the GUI element. The sendKeys()

function available in the Windows Forms library is used to send the keyboard input to the focused GUI element.

Exception Handler: This component is responsible for handling one of the following exceptions that occur while executing a macro: 1) process latency caused by the underlying OS when it is heavily loaded makes itself unable to render the next target GUI element in time; 2) missing prerequisite operators in the recorded macro (Ex: visiting a URL before accessing the webpage, clicking on a tab in the ribbon interface of Word 2007 etc); 3) notifications or alerts from the application that block the interaction in the user interface. All the above cases make the macro replayer unable to find the next GUI element. The exception handler retries several times to avoid the latency issue and then hands control to the user via the exception presenter.

Macro Recommender: This component analyzes the user activity history on an on-demand basis and suggests macros for the user to create. The *Raw Operator Recorder* and the *GUI Element Extractor* are reused to keep track of the user activity on the PC, which is stored as a dump. As introduced in the previous section, the macro recommender uses a suffix tree to process the user activity, since a suffix tree is a well-known linear-time solution to the longest common substring problem. Figure 22 uses an example history of operators, say "ABCABCDABE" to show the steps in determining all longest-matching repetitive sequences using a suffix tree. Each node in the suffix tree represents a sequence of operators and also the number of occurrences of the particular sequence (e.g. the leftmost "C_2" in (1) represents that "ABC" appears twice). The Macro Recommender removes redundant patterns by filtering the longest-matching repetitive sequences from the found sequences. It removes redundant prefixes in (2) and redundant suffixes in (3) by leveraging the information stored in the suffix tree. The Macro Recommender keeps track of the discovered patterns and suggest the new patterns to the user for macro creation.

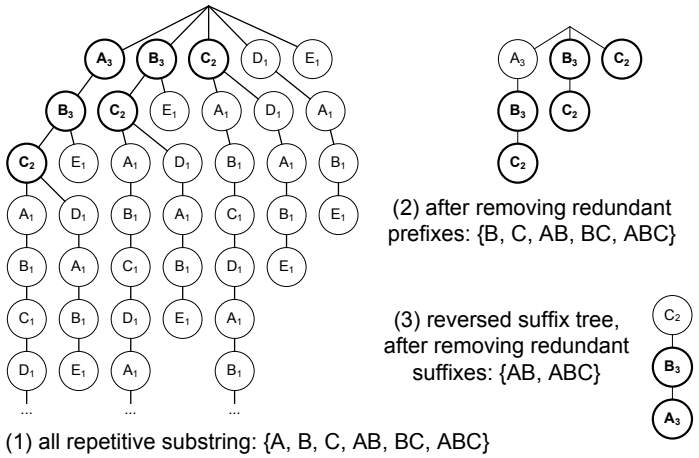


Figure 22: An example of macro suggestion with a suffix tree

Other Components: We now describe the other components that constitute the $\text{MORPH}_{\text{Aggregation}}$ server: 1) *Raw Operator Recorder* is involved in capturing the raw user input like mouse clicks and keyboard entries. 2) *Macro Repository* provides an interface to store and retrieve the recorded macros in the persistent database. 3) The *Macro Initiator* handles the replaying of macros on the desktop. It interfaces with the $\text{MORPH}_{\text{Aggregation}}$ client app and retrieves the macro from the repository when required and executes each operation involved in the macro using the *Operator Replayer*. 4) The *Parameter Handler* takes care of replaying parameter operations on the desktop. The handler provides the bounding box and a default value for the parameter, and it applies the user input to the GUI element.

5.4.2 $\text{MORPH}_{\text{Aggregation}}$ Client on Smartphone

The $\text{MORPH}_{\text{Aggregation}}$ client contains all the components of the Macro Presentation functional block and some components of the others. We prototype the solution by modifying AndroidVNC, an open-source VNC client. We modify only two files of the AndroidVNC source namely VncActivity, the main Activity object, and VncCanvas, the main GUI object of VNC interface. All $\text{MORPH}_{\text{Aggregation}}$ communication from the smartphone to the desktop is asynchronous and bidirectional. We now explain

Figure 23: $\text{MORPH}_{\text{Aggregation}}$ client screenshot

the different components:

Overlay Panel: The *overlay panel* is a control panel that is embedded into the remote computing client and provides a set of buttons to the smartphone user. Figure 23 shows a screenshot of the overlay panel and the underlying VNC client. There are five sets of buttons shown in the control panel depending on the stage of macro execution: application menu, macro menu, macro execution menu, parameter menu, and exception handling menu. The overlay panel shows only one menu at a given time and the panel added to the remote computing client using a layout that allows overlapping. The switching between menus is realized by toggling the visibility of the previous and the current menus. The overlay panel is added to the content of the VncActivity object with a FrameLayout that overlaps the panel on top of the VNC client. Since Android only allows the UI thread to modify the UI, other components that want to update the overlay panel need to go through the *UI updater*, a component created by us in the UI thread.

Macro Presenter: This component maintains the application menu and the macro menu, and it also responds to the selection of the user. It connects to the macro repository to retrieve the list of applications and the list of macros associated with a specific application. The applications and the macros are alphabetically ordered so that the users can easily find the macros. When an application is selected by the user,

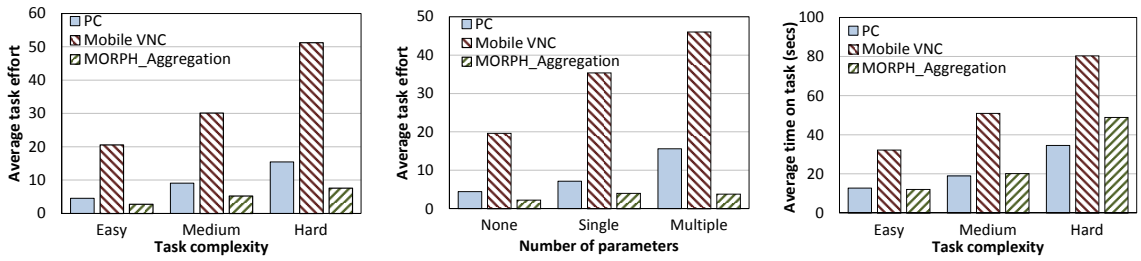
the macro presenter sends a request to the macro initiator to bring the application to the foreground. When a macro is selected by the user, the macro presenter sends a request to the macro initiator to execute the selected macro. Both requests are sent with asynchronous communication so that the UI of the smartphone is not blocked. While a macro is being executed by the macro initiator, the macro presenter shows the macro execution menu that allows the user to control the timing of the execution. As we have discussed in Section 5.3.4, the timing control allows the user to extend the recorded macros with new operators in runtime.

Parameter/Exception Presenter: This component handles the presentation of parameters and exceptions to the user. While receiving a notification of a parameter from the parameter handler, this component first shows the parameter menu in the overlay panel via the UI updater. Then it zooms the user to the bounding box of the parameter. This is done by using the zooming and panning functions in the interface provided by the underlying remote computing client, which is VncCanvas in our prototype. The response from the user could be a new value or selecting a default value provided by the parameter handler. Lastly, the presenter accepts the response from the user and returns it to the parameter handler to proceed with the execution. Similarly, the parameter/exception presenter shows the exception menu upon an exception notification from the exception handler on the desktop side. The exception menu contains three options namely "Retry", "Ignore" and "Abort". These options are self-explanatory.

Notification Receiver: This component receives notifications of new macros from the macro generator and gives updates to the user in the overlay panel.

5.4.3 Portability to other Platforms

Portability to other PC platforms: Since Mac OS and Linux also have their own accessibility frameworks [16, 13], the $\text{MORPH}_{\text{Aggregation}}$ solution can be easily ported



(a) Task effort by complexity (b) Task effort parameter type (c) Time on task

Figure 24: Overall performance enhancement with $\text{MORPH}_{\text{Aggregation}}$

to other PC platforms as well.

Portability to other smartphone platforms: The $\text{MORPH}_{\text{Aggregation}}$ solution can also be ported to other smartphone platforms such as iPhone, RIM, Windows Phone 7, Symbian, PalmOS as long as the source code of the target remote computing client is available. Note that the integration required with the client is very simple.

5.5 Performance Evaluation

In this section, we present the performance evaluation of our implementation of $\text{MORPH}_{\text{Aggregation}}$ with experiments involving real users.

Prototyping: Our prototype testbed consists of 1) a Dell desktop running Windows XP SP3 with a Pentium-4 2.8 GHz CPU, 3GB RAM, and a 19-inch monitor (1280x1024) and 2) a Samsung Galaxy S smartphone running Android 2.1 with 1GHz CPU, 512MB RAM, and a 4-inch screen (800x480). We evaluate our solution with nine Windows applications and six tasks of different complexity for each application. The full list of tasks and applications is shown in Table 7. While the tasks are pre-determined, they are designed from real-user activity dumps we collected for the motivation results used in Section 5.2.2. Since we observe that users not only use easy-to-find GUI elements but also those hidden inside layers of menus, we define the tasks with different levels of complexity to fully capture typical user activity. We use the AndroidVNC [1] as the VNC client for comparisons. Both the AndroidVNC and $\text{MORPH}_{\text{Aggregation}}$ apps are installed in the Android phone which connects to the PC

via a local Wi-Fi network. In the rest of the section, we refer to AndroidVNC as simply VNC.

Metrics: We compare the user experience of $\text{MORPH}_{\text{Aggregation}}$ with that of VNC and direct PC access using two objective performance metrics, time on task and task effort. Our definition of task effort only focuses on mouse/keyboard operators that can be identified and automated by a software system. It can be considered as a limited version of the Keystroke-level Model in GOMS family [49], which contains mental operators such as mental preparation and concentration shift that are not directly related our solution. We also get subjective feedback from real users and provide a CPU and memory profiling analysis of VNC and $\text{MORPH}_{\text{Aggregation}}$ to show the overheads introduced by our solution. Finally, we provide statistics from the offline macro suggestion for the user dumps collected from ten users and the potential task effort reduction for these users using the $\text{MORPH}_{\text{Aggregation}}$ solution.

Experimental methodology: We invited twenty-two volunteers in our experimental evaluation, and all of them are students whose age is between 20 and 30. While some of them were not smartphone users, all of them actively use their PCs for daily tasks. In the experiment, each user was randomly given two applications from the nine applications, and was asked to perform three tasks (one in each complexity category) in each application. While using $\text{MORPH}_{\text{Aggregation}}$, the smartphone had been loaded with the pre-defined 54 macros, and the user is asked to perform the six tasks with the corresponding macros. Since a user might not be familiar with the application or the smartphone, we asked the user to practice the tasks until they feel comfortable to perform experiments so that the learning effect is reduced. We perform within-subject evaluation (the user performs the same tasks on PC, on AndroidVNC, and then on $\text{MORPH}_{\text{Aggregation}}$) so that the users can give us their subjective feedback with side-by-side comparison. It should be noted that the experimental results here are only applicable to the scenarios where an experienced user has created macros

for her routine tasks, and she uses the macros to reduce her time and effort in doing these routine tasks from a smartphone. We use the traces from real users to evaluate the achievable time/effort reduction in accomplishing generic tasks using remote computing from a smartphone. We do not study the learning effort of using $\text{MORPH}_{\text{Aggregation}}$ and defer it as part of our future work.

Table 4: Task list and macros used for $\text{MORPH}_{\text{Aggregation}}$ evaluation

App	Complexity	Tasks	
Word	Easy	Open and print a file	Change format of a line
	Medium	Justify the entire text	Add border to document
	Hard	Insert picture & effects	Justify text & add a footer
Excel	Easy	Open and print a file	Sort data by a column
	Medium	Insert a bar chart	Insert a formula in a cell
	Hard	Import data from a csv file	Insert a scatter chart
Power Point	Easy	Add a picture in a slide	Change template for slides
	Medium	Print handouts of slides	Select an animation scheme
	Hard	Add a footer	Edit the master slide
Outlook	Easy	Print the current calendar	Print contact list
	Medium	Arrange mail	Arrange contacts
	Hard	Change email options	Change calendar options
Quicken	Easy	Print a summary report	Banking summary
	Medium	Export cash flow report	Add a transaction
	Hard	Compare spending by year	Find spending on clothing
IE	Easy	Print a webpage	Go to a specific webpage
	Medium	View an RSS feed	Check weather
	Hard	Change email options	Change tab options
Share Point	Easy	Add a new announcement	Upload a new document
	Medium	Add a new event	Add a new task
	Hard	Edit permissions	Check in a document
Visio	Easy	Print a file	Show a category of shapes
	Medium	Change pattern of a shape	Add shadow to a shape
	Hard	Configure layout	Flip a figure
Project	Easy	Print a Gantt chart	View network diagram
	Medium	View a resource sheet	Sort event list by criterion
	Hard	Create a specified report	Print multiple views

Measurement and analysis: During each experiment, we collect objective performance data with measurement tools built for both the PC and the Android phone. The tools measure both the task effort and the time on task for the user to execute a task. In VNC and $\text{MORPH}_{\text{Aggregation}}$, the measurement tool is integrated into the

menu of the Android app. In PC, the measurement tool is a stand-alone program that can capture raw user input to any application. The users provide their subjective opinion of using PC, VNC, or $\text{MORPH}_{\text{Aggregation}}$ in executing each task. We use Google Docs to create an online survey form where they can give their subjective opinion in an anonymous manner. We use averages to provide overall performance evaluation, but we don't provide confidence intervals since the grouped tasks contain different tasks that may belong to different applications or complexity.

5.5.1 Overall Performance Improvement

The performance results in Figures 24(a) show that our primary design goal of reducing task effort is achieved across different task categories. As discussed in Equation 38 in Section 5.2 task effort on VNC contains a component of task effort on the PC and a component of mobile inflation. $\text{MORPH}_{\text{Aggregation}}$ reduces both the components as evidenced by a lower task effort than both the effort on VNC and the effort on PC. Figure 24(b) shows the task effort categorized in terms of number of parameters required for the task. We observe that with more parameters, the effort reduction ratio is higher since the users can significantly reduce their effort by using default values. Figure 24(c) shows a significantly lower time on task for $\text{MORPH}_{\text{Aggregation}}$ when compared to those of VNC for all three task categories. Also, we observe that $\text{MORPH}_{\text{Aggregation}}$ takes almost the same average time as working on a PC. While for some tasks, $\text{MORPH}_{\text{Aggregation}}$ takes a slightly lesser time than a PC because of the task effort reduction with operator aggregation. The time on task in the VNC is significantly higher since the users not only have to perform all operators, each of them is inflated due to the input constraints in the smartphone.

5.5.2 Performance Improvement by Application

While we show average results in the previous subsection, Table 5 shows the reduction of time on task when using $\text{MORPH}_{\text{Aggregation}}$ when compared to VNC for individual

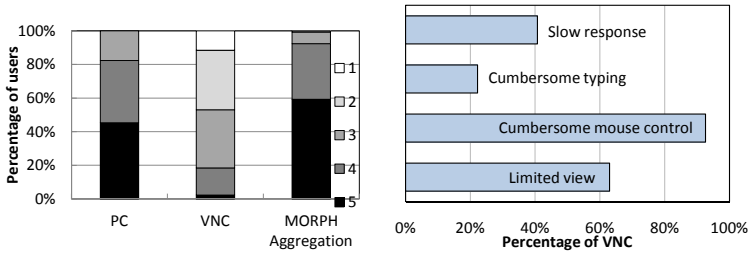
Table 5: Time on task and reduction percentage with $\text{MORPH}_{\text{Aggregation}}$

Application	VNC	$\text{MORPH}_{\text{Aggregation}}$	Reduction
Word	77.22	16.07	79%
Excel	59.10	25.63	57%
PowerPoint	50.53	21.04	58%
Outlook	41.75	8.04	81%
Quicken	105.43	24.81	76%
Internet Explorer	24.66	21.30	14%
Visio	38.96	13.33	66%
Project	41.93	10.19	76%
SharePoint	53.86	31.56	41%

applications. We observe that the time on task by $\text{MORPH}_{\text{Aggregation}}$ to perform tasks reduces for all applications. The applications have different types of GUI menus ranging from the traditional menus with a deeper structure for Quicken to the newest *Ribbon interface* for MS Office 2007 products. We observe that irrespective of the GUI menu type, VNC requires a lot of time for users to navigate the menu for performing tasks. The time reduction varies from 14 % for IE to 81 % for MS Outlook.

5.5.3 Subjective Opinion

After performing all experiments, we ask the users to provide their subjective opinion on using different platforms to accomplish tasks. We ask the user "How would you rate your user experience in performing the task using certain platform?" with a scale value from 1 (poorest) to 5 (best), which is known as the Likert scale [61]. Figure 25(a) shows first a significant decrease in the subjective evaluation for VNC when compared to PC, and our solution provides a significant increase back to the PC-level. Everyone in the focus group rated $\text{MORPH}_{\text{Aggregation}}$ greater or at least equal to VNC for every task performed. In fact some users rated $\text{MORPH}_{\text{Aggregation}}$ higher than a PC because our solution reduces the effort of performing operators on the smartphone to less than even that of performing on a PC. Figure 25(b) shows the main reasons of users' frustration on using VNC from the smartphone, and most of



(a) Likert scale (b) Frustration on VNC

Figure 25: Evaluation of subjective opinion on $\text{MORPH}_{\text{Aggregation}}$

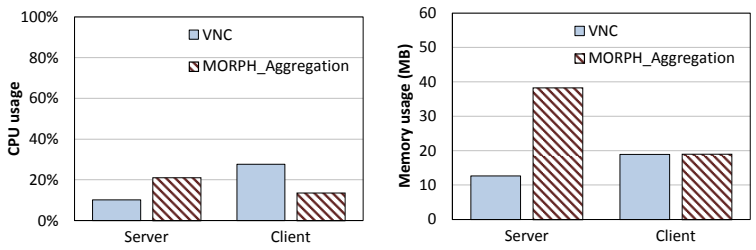
them are related to the interface constraints and increased task effort.

5.5.4 Overhead Analysis

We now present a CPU and memory usage profiling of our solution. We collect the CPU and memory statistics while executing a hard task of MS Word on both the PC and the smartphone. The statistics for PC are collected using perfmon, a system monitor utility tool provided by Microsoft along with Windows. The statistics for the Android smartphone are collected using SystemPanel, one of the best-rated system monitor apps in the Android Market. Figure 26(a) shows the average CPU usage of VNC and the $\text{MORPH}_{\text{Aggregation}}$ solution at the desktop and at the smartphone. The results of $\text{MORPH}_{\text{Aggregation}}$ on the PC include the unmodified VNC server running on the PC. The $\text{MORPH}_{\text{Aggregation}}$ server takes 7.19% atop the unmodified VNC server [2]. The integrated client takes less CPU than the unmodified VNC client, and it can be attributed to the reduced load at the smartphone side due to less user interaction with the app.

Figure 26(b) shows the average memory usage of VNC and $\text{MORPH}_{\text{Aggregation}}$ at both server and the client sides. The memory usage measurement is based on the unique set size in Android and the private bytes in Windows, since both indicate the memory size exclusively allocated to the process. The $\text{MORPH}_{\text{Aggregation}}$ server

[2]The CPU usage of the unmodified VNC server is also increased by 3.71% when running with $\text{MORPH}_{\text{Aggregation}}$ since operators are executed at a faster rate.



(a) CPU usage (b) Memory usage

Figure 26: Overhead analysis of $\text{MORPH}_{\text{Aggregation}}$

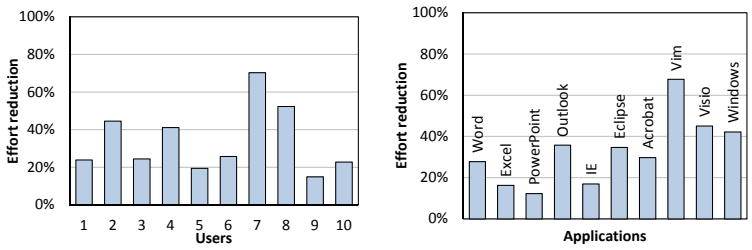
Table 6: Statistics from offline macro suggestion

User ID	Number of tasks	Average length	Average daily freq.
1	143	2.34	2.96
2	652	2.77	0.47
3	1125	2.87	1.11
4	471	7.28	2.20
5	156	14.18	3.61
6	59	5.24	1.78
7	100	1.96	1.32
8	53	1.72	0.84
9	493	2.68	1.24
10	282	17.58	5.57

takes about 26MB atop the unmodified VNC, and the overhead is acceptable since current PCs typically have a large memory. The $\text{MORPH}_{\text{Aggregation}}$ client takes only 100KB more memory than the unmodified VNC client. Since we have designed the $\text{MORPH}_{\text{Aggregation}}$ solution to have most of the processing on the server, the client is very lightweight and efficient.

5.5.5 Results of Offline Macro Suggestion

Table 6 shows the statistics from running the offline macro suggestion tool on the user activity dumps of the ten users we collected for Section 5.2.2. The table shows the number of tasks identified for aggregation on a per user basis. The table also shows the average number of operators for the tasks identified and the average frequency of repetition of the tasks. We observe that for most users, the tool is able to identify



(a) Effort reduction by user (b) Effort reduction by application

Figure 27: Trace-based evaluation of effort reduction with $\text{MORPH}_{\text{Aggregation}}$

more than one hundred repetitive tasks. Further we observe that these tasks are typically executed by the user at least once or twice per day. The average length of the tasks is at least two for most users with User 10 having an average task length of 17.58.

5.5.6 Trace-Based Evaluation of Task Effort Reduction

While we have shown task effort reduction for the specific pre-defined tasks, we now show the potential task effort reduction for realistic user activity. In a realistic scenario, it may not be feasible to optimize every operator that the user intends to perform from a smartphone. Thus, we base our analysis on the traces of user activity we collected from real users for evaluating the effort reduction achievable by $\text{MORPH}_{\text{Aggregation}}$ [3]. We use the following equation to estimate the effort of executing a task with $\text{MORPH}_{\text{Aggregation}}$:

$$TaskEffort^{MORPH_{Aggregation}} = min(\, TaskEffort^{PC} \times Reduction, 1)$$

where *Reduction* is the effort reduction from *PC* that we observe in the experiments, which value is 0.61. Note that the task effort of executing a macro is at least one. In the user activity history, we only match with operator sequences that have at least

[3]Note that the trace-based analysis is performed by assuming that the typical user activity on a smartphone will be the same as that on a PC. In reality, this might not be the case and the users might actually want to perform only a subset of tasks performed on a PC. $\text{MORPH}_{\text{Aggregation}}$ will reduce task effort to much greater extent than our pessimistic estimate.

two operators. This fits with the minimum macro length in the task list and makes the analysis more realistic. Figure 27(a) shows the effort reduction achieved with $\text{MORPH}_{\text{Aggregation}}$ for each of the users. The reduction ranges from 14.85% to 70.30%, and the average of 37.71% shows that $\text{MORPH}_{\text{Aggregation}}$ can provide significant effort reduction in real user activity. Similarly, $\text{MORPH}_{\text{Aggregation}}$ reduces task effort in each of the top applications used by the users, as shown in Figure 27(b).

CHAPTER VI

ENABLING RAPID MOBILIZATION FOR ENTERPRISE APPLICATIONS

6.1 Overview

In the previous chapter, we have presented the aggregation transformation service of *MORPH* protocol that reduces the effort and time required to accomplishing a repetitive task in remote computing. In this chapter, we present the design and implementation of the other two core transformation services of *MORPH*. First, translation transformation service presents a smartphone friendly view by converting each element in the *virtual view* into a native element in the smartphone. It reduces the task effort and time-to-task even for non-repetitive tasks. Second, traffic suppression transformation service intelligently suppresses and reduces traffic required for the remote access leveraging knowledge of the *virtual view*. Using *MORPH* as the underlying protocol, we present a solution of rapid application mobilization called **Mobile* that integrates the two core transformation services. We prototype the solution and demonstrate significant performance gains using user-experiments. While we focus on enterprise applications since mobilizing them is more challenging, the solution is also applicable to consumer applications.

6.2 Solution Basics and Design Elements

**Mobile* is an application mobilization solution that is built on *MORPH*. At the high level, the **Mobile* backend resides on the PC and offers an "optimized window" into the backend application for the smartphone user to view and control the application. Figure 28 illustrates the network architecture of application mobilization with remote

computing from a smartphone. The **Mobile* server runs on the backend PC, which communicates with the **Mobile* client runs on the smartphone using the *MORPH* protocol. The client receives updates to the application views from the server, and renders them. The client sends back any user-input, and the server executes the input on the PC. The **Mobile* client, as shown in Figure 29, is a single native app provided to the smartphone user that contains multiple app-launchers one each per applications installed on the PC.

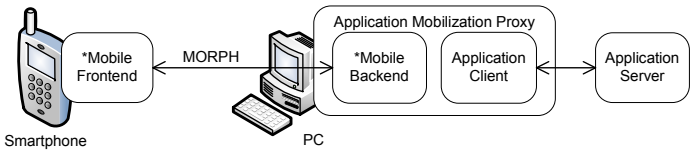


Figure 28: Application mobilization with remote computing

Briefly, the "optimized window" consists of providing a *dynamic UI transformation* to make it more amenable and appropriate for the smartphone. For example, mouse clickable tabs could be transformed into touchable buttons, and a pull down menu could be transformed into a spinner wheel. It also consists of *reducing the remote computing traffic* by intelligently suppressing and compressing the remote view information with knowledge of the smartphone environment.

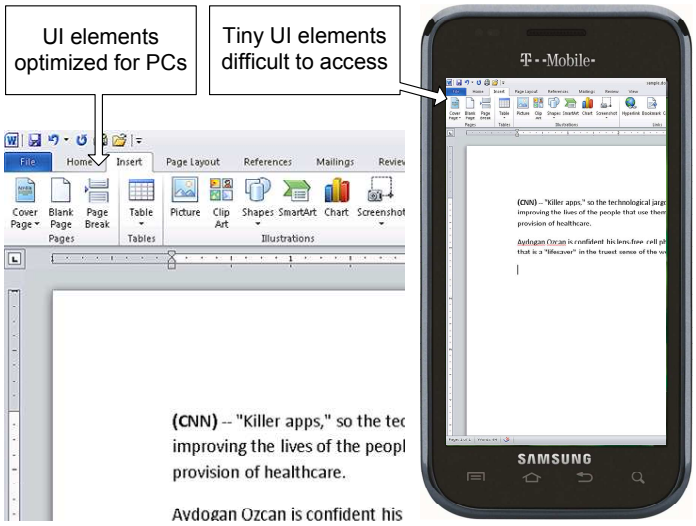
For ease of exposition, we also consider a baseline version of **Mobile* that does not have the aforementioned design elements and instead simply provides a thin client window into the PC. We use the baseline to set up the motivation for the design elements and also later to compare performances. While we use VNC as the underlying thin client solution for **Mobile*, the design elements and concepts presented may be easily extended to other thin clients such as RDP.

Figure 30(a) shows part of a screenshot of MS Word on a PC platform, and Figure 30(b) shows the corresponding screenshot of the baseline thin client in an Android smartphone (Samsung Galaxy S).

Figure 29: Screenshot of app-launchers in the **Mobile* client

6.2.1 Dynamic Interface Transformation

**Mobile* is designed with the capability to perform *dynamic interface transformation* that dynamically, and in real-time converts the user interface of the PC application into a user interface appropriate for the smartphone. The noteworthy aspects of the transformation design are the following: (i) Unlike traditional thin client solutions that tap into the view on the backend at the pixel or graphical primitives level, **Mobile* taps into the view at the level of UI elements. This higher level of abstraction at which **Mobile* learns of the view allows it to construct a common UI description of the same. The common UI description is then shipped to the smartphone where the view described is then rendered using elements available in the smartphone's UI library thus making the view more tailored and useable. Note that this approach of decoupling the abstraction and rendering allows the same common UI description to be rendered differently on different smartphone platforms. On Windows PCs, the UI element information can be extracted using the UI automation library [28]. (ii) Since the transformation is performed dynamically and automatically, it is possible that **Mobile* encounters UI elements that it does not recognize or cannot get sufficient



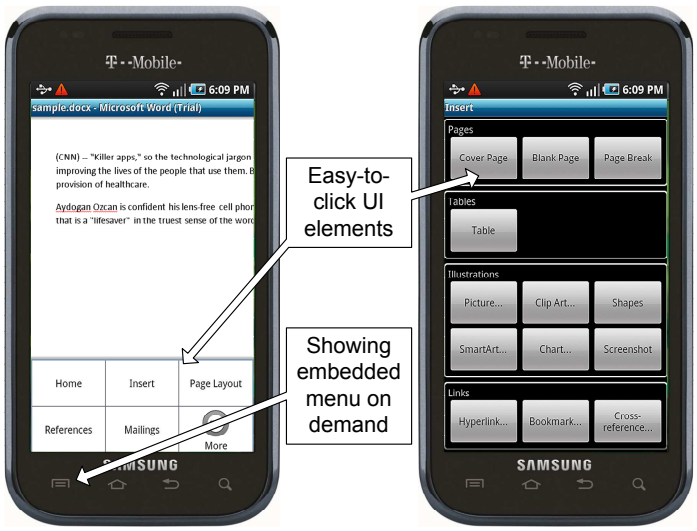
(a) PC interface (b) Baseline interface

Figure 30: Screenshots of MS Word in PC, baseline

information about. To tackle such scenarios **Mobile* employs a robust fallback strategy of transferring and rendering unrecognized UI elements using the underlying thin client mechanism. Thus, **Mobile* is capable of rendering a single application view that is a composite of transformed and untransformed UI elements. An example of a UI that is opaque to **Mobile* and hence has to be rendered as-is is the workspace in Microsoft Word. Screenshots of the **Mobile* transformed UI (implemented for the Android OS) for MS Word are shown in Figures 31. We can see how the different UI elements are more accessible for users to navigate through.

6.2.2 Traffic Optimization

One of the obvious dependencies **Mobile* has is that of network connectivity. While it is a definite drawback in comparison to a *standalone* traditionally mobilized application, we argue that many if not most applications in an enterprise setting do operate in a client server mode and hence connectivity is a requirement imposed even by the underlying application. Of the three applications we consider, both Microsoft Project and Intuit Quickbooks rely on connectivity to varying degrees while Microsoft Word does not require any connectivity.



(a) Transformed window (b) Transformed Toolbar

Figure 31: Screenshots of MS Word in *Mobile

Thus, **Mobile* is designed with the capability to intelligently optimize the traffic incurred by remoting. There are two design elements that constitute this capability: (i) Thin client solutions by default perform *short-term lossless compression* of the view to be rendered. In other words, a particular view is differentially coded with respect to the last view. However, we posit that in a smartphone environment the different views of an application are likely to repeat on an exact basis due to the lack of dynamics such as window re-sizing re-positioning. Thus **Mobile* performs *long term inter-view differential coding* in addition to the short-term compression for UI elements that are conveyed using the traditional thin client scheme. The client maintains a dictionary of past views per application, and the **Mobile* proxy performs its differential coding with the optimal past view and appropriately informs the client the index of the past view to use for the decoding. Note that the dictionary could either be learned or pre-loaded. (ii) In addition to better compression of the views, **Mobile* also intelligently suppresses traffic when updates in the server side view are not likely to be visible at the client side. Examples of such scenarios include when local rendering of UI elements blocks the visibility of updates and updates that occur

outside the bounding box of the visible application window. For example (of the former scenario), when showing the transformed UI of a toolbar as in Figure 31(b), any updates to the workspace is not necessary to be conveyed.

6.3 Solution Details

In this section we present the solution details for **Mobile*. The **Mobile* client on the smartphone contains a thin client frontend. The **Mobile* backend contains a corresponding thin client server to enable the baseline remote computing. Figure 32 shows the different components that achieve dynamic interface transformation and traffic optimization.

As introduced earlier, the **Mobile* backend resides on the PC and interacts with the various applications installed on the PC. In the rest of the section we first explain the individual components involved in each of the three design elements. While the overall design of **Mobile* can be applied to any combination of PC platform, mobile platform, and thin client computing technology, we ground the discussion with the implementation details of our prototype that uses Windows OS as the PC platform, Android OS as the mobile platform, and VNC as the thin client computing technology. While we present details of a Windows OS based prototype that uses the Microsoft UI Automation framework, we note that the solution is feasible for other OSes such as Linux and MAC OS, using similar accessibility frameworks [17, 14].

6.3.1 Dynamic Interface Transformation

We first explain the *common UI description* used to transform the PC UI to the smartphone UI. The user interface is described using a tree data structure of a single root object and several children objects. The children might in turn have more children objects. Each object can represent UI control types such as window, menu, dialog, button, check box, etc. Dynamic interface transformation is realized with three functional blocks: Proxy UI processing maps each PC UI element to an object;

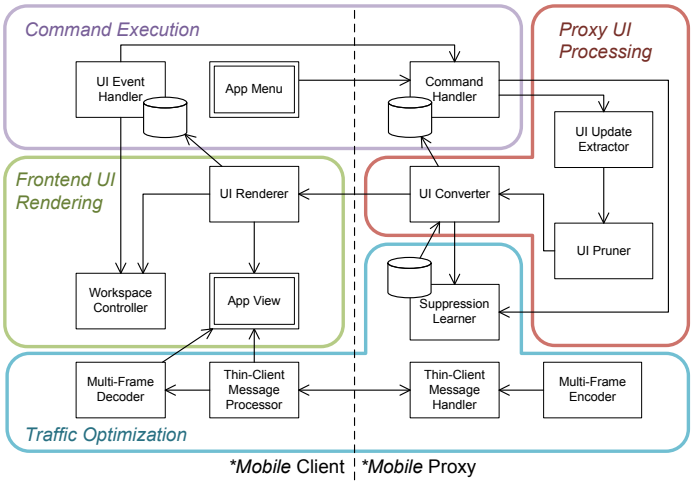


Figure 32: Components in the core **Mobile* system

(a) Project dialog (b) Transformed dialog (c) QuickBooks menu (d) Transformed menu

Figure 33: Screenshots of dynamic interface transformation

frontend UI rendering renders each object as a native smartphone UI element; command execution reversely maps smartphone elements to PC elements to perform user interaction.

Proxy UI Processing: This functional block is responsible for extracting the uniquely identifiable information of the various PC UI elements and converting them to the common UI description. We denote a "view" as a set of PC UI elements that will be rendered together in a single page on the **Mobile* client. A view could denote a main window of an application, a drop-down menu, or a separate dialog.

The *UI update extractor* extracts all the UI elements belonging to current view shown in PC. The current view is identified by events that are triggered automatically by the OS or generated by other components of the solution. In our implementation, the extraction process itself is achieved by using the TreeWalker object provided by the UI Automation library.

There are often a few UI elements of a PC application, in a given view, that are unnecessary for the **Mobile* client. For example, PC elements such as buttons for maximizing/minimizing a window are unnecessary on the **Mobile* client. There are also some redundant UI elements that serve as containers for other UI elements that are also unnecessary for the **Mobile* client. The *UI Pruner* helps in eliminating such unnecessary UI elements from the common UI description.

Once all the PC UI elements of a given view are extracted and pruned, the *UI converter* creates a common UI description for the entire view with the necessary tree data structure of objects. The common UI description of the view is also sent to the **Mobile* client for rendering on the smartphone.

Frontend UI Rendering: The *UI renderer* is the core component in the **Mobile* client that performs the rendering of the transformed user interface. We design the rendering process of the transformed UI elements with a push model. Each view of a common UI description is rendered as a separate full-screen page in the **Mobile* client. Each type of object is mapped into a native UI element in the target mobile platform. When there is no direct mapping of an object in the target mobile platform, we use an element with similar functions or develop a priori a custom element. For example, "combo box", a PC UI element does not have a direct equivalent in Android, and hence it is transformed into a "Spinner", which pops up a selection list upon a click. Transformation of window and Ribbon toolbar is shown in Figures 31(a) and 31(b). Figure 33(b) shows the transformation of the Font dialog in MS Project. Figure 33(d) shows how a pop-up menu in QuickBooks is transformed into a scrollable ListView

in Android.

The *workspace controller* seamlessly renders the workspace on the **Mobile* client and also sends user actions on the workspace such as mouse-clicks and keyboard entries to the thin client message handler on the **Mobile* proxy. Several mechanisms are added to improve usability: 1) When the user clicks on an editable element in the workspace, the controller automatically opens the software keyboard for the user. 2) The controller also remembers the current position and zoom scale in workspace between view switching to provide seamless operation for the mobile user.

Command Execution: This functional block is responsible for understanding user intent on the **Mobile* client and initiating the action required to fulfill the user intent. When the user clicks on a UI element or enters some text in an edit box on the app view, an "element-click" or a "set-text" command is sent to the **Mobile* proxy respectively. The *UI event handler* maps the UI element to the corresponding object and sends it to the *command handler*, which maps it to the UI element reference to perform the necessary action on the PC. While the UI converter and the UI renderer perform transformation to and from the common UI description, they store the mapping into a hash map, and both the UI event handler and the command handler use the hash maps for quick retrieval. In Windows, APIs provided by UI Automation are used for performing the actions.

The command handler is also responsible for opening applications. Upon launching of the **Mobile* client, the *app menu* loads the profile of the mobile user and shows the list of applications available for use. When the user clicks on an application icon, an "application-open" command is sent to the *command handler* residing on the **Mobile* proxy. The command handler initiates a new instance of the application if not already opened or just brings the application to the foreground. The command handler then sends an event to the UI extractor asking it to extract the UI information of the main window.

6.3.2 Traffic Optimization

Traffic optimization is an important design element in **Mobile* to reduce the network traffic usage by the thin client solution for the workspace. Since we rely on VNC as the underlying thin client solution in our prototype, we first provide a primer on VNC. VNC [27] is based on the open source Remote Framebuffer (RFB) protocol [26]. The screen update is performed with a polling mechanism. The VNC client requests for updates of the desktop screen of a remote PC where the VNC server resides in. The server pulls the framebuffer, which is the complete pixel-by-pixel information of the current video display. The VNC server compares the current framebuffer against the previous one, and it sends only an incremental update that contains the delta-difference from the previous framebuffer it has already sent to the client. Once the VNC client receives a framebuffer update, it refreshes the local screen display with the updates and requests for the next update. We now explain the details of traffic optimization using the following functional blocks:

Reduced update scope: **Mobile* reduces the scope of screen updates in the thin client computing to eliminate unnecessary information in the space domain. Unlike traditional thin client solutions that displays the full desktop screen, only the workspace of the application needs to be displayed using remote computing in **Mobile*. A normal VNC client always requests for the entire screen window as the scope of any updated content. The request message is modified by the *thin client message processor*, which uses the bounding box of the workspace as the scope for update request.

Request suppression: **Mobile* further eliminates unnecessary traffic in the time domain. It suppresses not just screen updates from the thin client server but also the requests from **Mobile* client when it is not displaying the workspace. For example, when menus/ dialogs are opened and fully transformed, the common UI description

contains only transformed objects but not the workspace, and thus thin client computing is not needed in those views. To enable this suppression, we use a learning algorithm to identify these scenarios. We note that a menu/dialog is opened only when some UI element is clicked on the PC application. When a menu/dialog is opened for the first time, the *suppression learner* associates the menu/dialog view to the last UI element clicked. When the same UI element is transformed the next time around, the suppression flag is filled for the corresponding common UI description object. To enable the actual suppression mechanism in the VNC, we define a new message in the RFB protocol called "UpdateSuppress". The "UpdateSuppress" message instructs the *thin client message handler* to prevent updates to the previous update request received from the VNC client. When a rendered UI element that carries a suppression flag is clicked on the **Mobile* client, the message processor is notified by the *UI event handler* to perform update suppression. The message processor sends an "UpdateSuppress" message and holds the new updates generated by the VNC client. When the UI renderer later shows a view with the workspace, the message processor sends the a regular framebuffer "UpdateRequest" message to make the VNC server enter its normal state, where it can send regular updates of the workspace screen.

Multi-frame encoding/decoding: As discussed earlier, **Mobile* leverages the *long-term* redundancy across multiple screen views to further reduce traffic consumption by the thin client computing. Both the VNC server and client maintain a consistent cache of multiple screen framebuffers that have been seen in the thin client connection. Whenever the server has to send a screen update to the frontend, it compares the current framebuffer with all the multiple screen framebuffers and identifies one where the delta-difference is the minimum. The server encodes the update and instructs the client to use the correct reference screen to apply the update. The consistency of the screen caches at both sides is maintained with a simple mechanism. When the server stores the current screen into the cache, it adds the cache

update information into the update message. In our implementation, we modify the "FramebufferUpdate" message in the RFB protocol to include the source index and a destination index in the screen cache. The screen cache is maintained with a standard cache replacement algorithm used in microprocessor cache replacement policies, such as first-in-first-out (FIFO) or least frequently used (LRU).

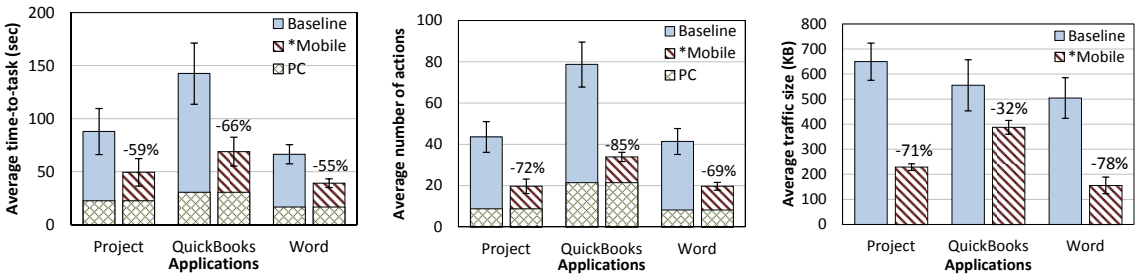
6.4 Performance Evaluation

In this section, we present the performance evaluation of our implementation of **Mobile*.

Prototyping: The client device is a Samsung Galaxy S smartphone running Android 2.1 with a 1GHz CPU and 512MB RAM. The application proxy is an Windows Server 2003 instance on the Amazon EC2 cloud. The **Mobile* prototype is built by modifying the open-source AndroidVNC [1] at the frontend and TightVNC server [27] at the backend. We study three enterprise applications namely Quickbooks Enterprise Solutions, MS Project and MS Word. Of the three, Quickbooks is a server-client based application while the other two are standalone PC applications.

Metrics: The primary performance metrics that we study include time-to-task (time taken to complete a pre-defined task), number of actions (clicks/keyboard entries) to complete a task, and traffic consumption in accomplishing tasks of enterprise applications from a smartphone.

Methodology: We rely on two sets of experiments for the results presented in the section. For the first set of results, we rely on an user-study with ten volunteers invited from both academia and industry. The volunteers were picked to represent diversity across age-groups (25-40), gender and employer type (large university, start-up, large enterprise). Eight of the volunteers were heavy smartphone users. Each user was asked to perform one representative task for each of the three applications (the bold faced tasks in Table 7) and using all three interfaces (PC, smartphone



(a) Time-to-task (b) Number of actions (c) Traffic consumption

Figure 34: Performance of the mobilized applications in the Wi-Fi network

baseline, smartphone **Mobile*). We measure time-to-task, number of actions and traffic consumption as the metrics of interest. For the second set of "microscopic" results, we use two expert users to study (a) the sensitivity of the performance results across different types of tasks for each application and different network environments (Wi-Fi and 3G); (b) how **Mobile* compares against custom built applications; and (c) the **Mobile* response time and overheads.

6.4.1 Time-to-task

Figure 34(a) shows the average time taken in accomplishing a task of the applications from the smartphone (with 90% confidence interval). The time taken in PC is also shown in the Figure as the benchmark comparison. Since the user interface in the PC applications is already optimized, the time-to-task we can achieve in a mobilized app can only equal to or greater than that of PC. As shown in the figure, **Mobile* achieves 59%, 66%, and 55% of the achievable performance enhancement for each of

Table 7: Applications and Tasks

	Project	QuickBooks	Word
1	add a task	add a customer	format text
2	configure a task	add an item	**apply a watermark**
3	link tasks	add an invoice	add a page border
4	view network diagram	receive a payment	justify a paragraph
5	**assign a resource**	view unpaid invoices	add a numbered list
6	update task schedule	**send a sales report**	insert decorative text

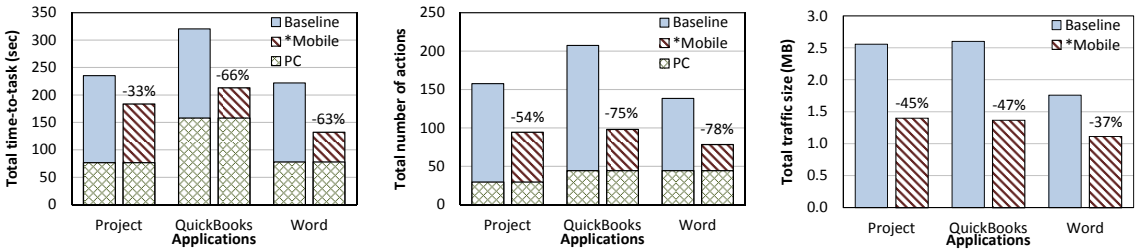
the three applications.

6.4.2 Number of Actions

Figure 34(b) shows the average number of actions required in accomplishing a task of the applications using the smartphone. Similar to the time-to-task, we also show number of actions required in directly using the PC. **Mobile* achieves 72%, 85%, and 69% of the achievable performance enhancement. In Microsoft Project and Word, the reduction in number of actions mainly comes from the transformation of the Ribbon toolbar. Since the toolbar is designed to be manipulated with a mouse instead of a touch pad, some of the buttons have a very small size. In the baseline thin-client solution, the users have to zoom/pan the screen several times to be able to accurately click on a button. By transforming the buttons into ones with an appropriate size, the Ribbon toolbar of Project and Word become easy to manipulate. On the other hand, the reduction in number of actions in QuickBooks is contributed by the transformed pop-up menus. The menus appear small and cluttered in the baseline thin-client solution, and **Mobile* transforms them into scrollable lists that are also used in native apps.

6.4.3 Traffic Consumption

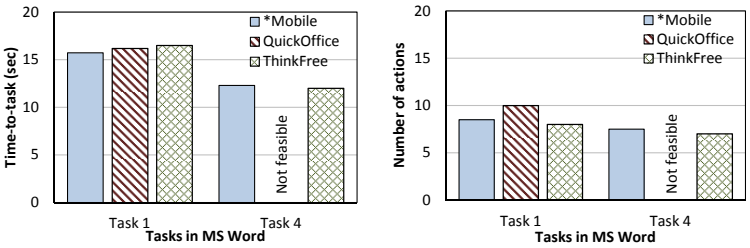
Figure 34(c) shows the traffic consumption for completing the tasks. The traffic optimization in **Mobile* is able to reduce the traffic size by more than half in Microsoft Project and Microsoft Word, and about one third of traffic is reduced in Intuit QuickBooks. The reason of the better reduction ratio in Project and Word may be attributed by the higher redundancy existing in the usage of the two applications. Common operations in the two application cause switches between different views and this results in frequent redundant updates. For example, users switches between different Ribbon tool bars when using both applications. Here the multi-screen buffering algorithm helps in reducing the traffic. While menus in QuickBooks



(a) Time-to-task (b) Number of actions (c) Traffic consumption

Figure 35: Performance of the mobilized applications in accomplishing a task set in the 3G network

Task feasibility	*Mobile	QuickOffice	ThinkFree
Format text	O	O	O
Apply a watermark	O	X	X
Apply a page border	O	X	X
Justify a paragraph	O	X	O
Add a numbered list	O	X	X
Insert decorative text	O	X	X



(a) Task feasibility (b) Time-to-task (c) Number of actions

Figure 36: Comparison with manually-built mobile apps of Word

also cause redundant updates, those updates have a smaller size.

6.4.4 Sensitivity to tasks and network environments

We perform microscopic analysis by considering more extensive tasks and varying the access technology. Figure 35(a) shows the total time taken for accomplishing a task set (six tasks) of the various applications using the smartphone via its 3G network. Similarly, Figure 35(b) and Figure 35(c) show the effort and traffic consumption during the same set of experiments. While accomplishing a task set takes more time, actions, and traffic than a single task, **Mobile* consistently provide considerable performance improvement for all applications and tasks tested in the experiments. The performance using the Wi-Fi network is similar to that of 3G, and thus is not presented due to lack of space.

6.4.5 Comparison with Custom-Built Apps

While **Mobile* is a rapid solution for mobilizing applications, the traditional strategy to mobilize applications is a clean slate approach that develops the mobile app from scratch. Thus, it is pertinent to compare **Mobile* against such a solution. There are two ways in which such a comparison can be done. The first is to compare applications mobilized through the two approaches using metrics such as time to task and feature parity. The second is to compare, for the functionality supported by the **Mobile* mobilized app, how much comparative effort would be required using the traditional strategy. We perform the first comparison objectively, and provide qualitative arguments for the second comparison.

We now compare the performance of **Mobile* with custom built native apps objectively. We identify two equivalent native apps for MS Word known as Quickoffice and ThinkFree Office from the Android Market. However, these native apps do not implement the complete functionality of MS Word. Rather, they provide only a subset containing a few basic edit functions. Figure 36(a) shows the possibility of achieving the six tasks that we identify. We observe that Quickoffice can be used only for one task while ThinkFree Office can be used for two of the six tasks. On the other hand, **Mobile* can provide the complete functionality of MS Word thus enabling all six tasks. Figures 36(b) and 36(c) show the two metrics of time and number of actions for the possible tasks using the native apps. We observe that the performance of **Mobile* is comparable to the native apps. Thus, we see that while providing significantly more functionality than custom built native apps, **Mobile* provides comparable performance in terms of usability.

The aforementioned lack of feature parity in custom-built mobile apps is closely related to the large cost to develop a full functionality app. As per mobilization vendors, the typical cost to mobilize an enterprise application ranges from a time effort of six months to a few years, and a cost ranging from $500,000 to $3000,000.

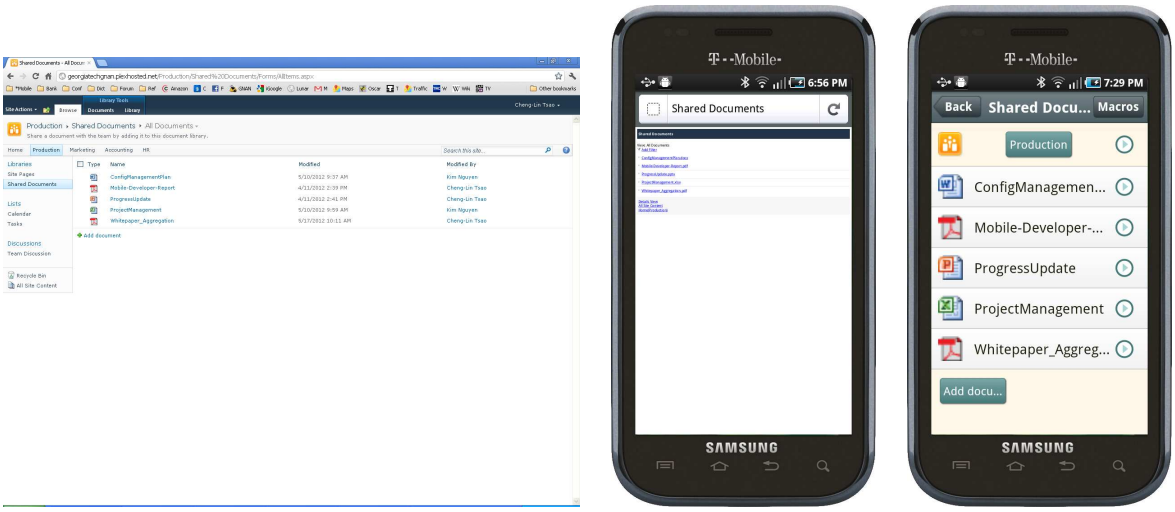
Comparatively, the time (and hence cost) to mobilize an application using **Mobile* is smaller by several orders of magnitude. We substantiate this claim by considering the time effort required to mobilize three typical enterprise applications, i.e. MS Project, Intuit QuickBooks, and MS Word. It takes approximately 17 minutes to mobilize each of the three applications using **Mobile*. The breakdown of the time requirement is listed in Table 8.

6.4.5.1 Web-based Enterprise Applications

Finally, it should be noted that mobilizing enterprise applications, even when they are web based, is prohibitively high when relying on the traditional clean slate strategy. This is somewhat in contrast to mobilizing web sites, which is a somewhat easier problem. Web sites, such as New York Times http://www.nytimes.com, focus on serving content to the web clients. Such web clients typically use a simple interaction model and have no application and business logic. Whenever they interact with the web site, an HTML document and its CSS style files are delivered to the client to describe the current view. Mobilizing web sites can be done by changing the HTML and CSS files require less effort, while still not trivial, to mobilize. On the other hand, web applications, especially enterprise web applications, typically provide rich functionality and may even contain part of the business logic residing at the client side. Such clients are typically based on client-side web technology such as AJAX where the presentation or functionality logic is hosted and executed in the web client. The client dynamically loads data and information from web servers, middleware servers,

Table 8: Time requirement in application mobilization using **Mobile*

Stage	Time spent
Instance initialization	10.2 min
Software installation	5.3 min
Security configuration	1.3 min
Mobile app deployment	0.4 min
Total	17.2 min

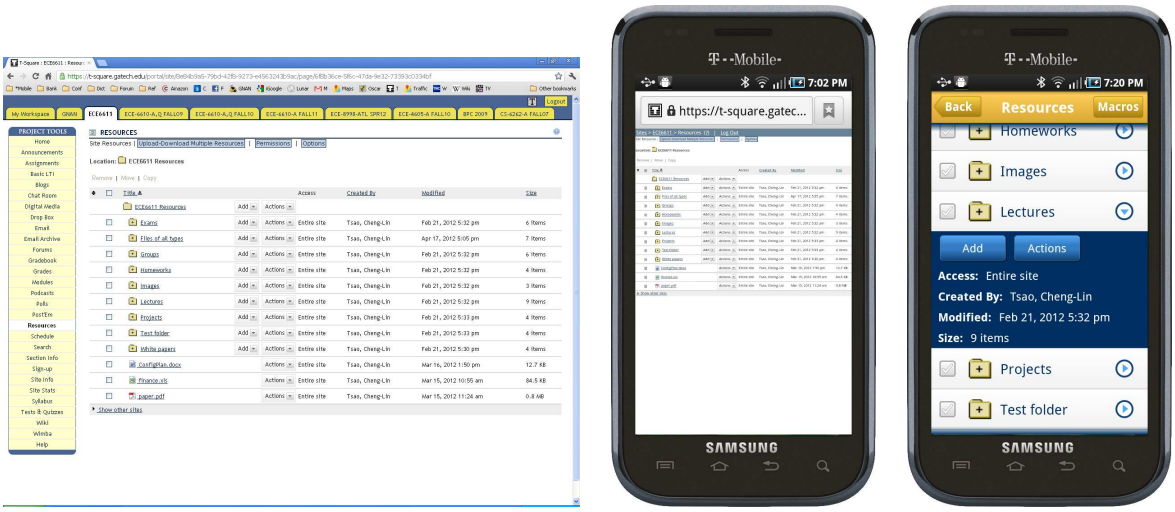


(a) PC version (b) Mobile version (c) *Mobile

Figure 37: Comparison of PC version, mobile version, and **Mobile* of Microsoft SharePoint

or database servers. As a result, mobilizing such web applications requires changes to the client-side code, which is JavaScript in AJAX clients, besides simply the changes in HTML and CSS files. Such code rewriting in mobilizing web application clients is similar to that of mobilizing native clients and thus takes significantly more effort than mobilizing web sites.

This latter point about web based enterprise applications is illustrated by considering two other enterprise applications MS Sharepoint and Georgia Tech T-square (a Sakai CLE based collaboration application). Because of the high cost in providing a smartphone-friendly view in web applications, the current mobile version provided in the two web applications either have a primitive interface or an interface very similar to the PC version. Figure 37(a) shows the screenshot of the shared documents in a site hosted in the full-blown PC SharePoint application. Figure 37(b) shows the screenshot of the same tool in the same site hosted in the mobile version of SharePoint. The mobile site has a very primitive interface that lacks the rich functionality provided in the PC site. Figure 37(c) shows the same SharePoint application mobilized by **Mobile*. Figure 38(a) shows the screenshot of the resource tool in a course



(a) PC version (b) Mobile version (c) *Mobile

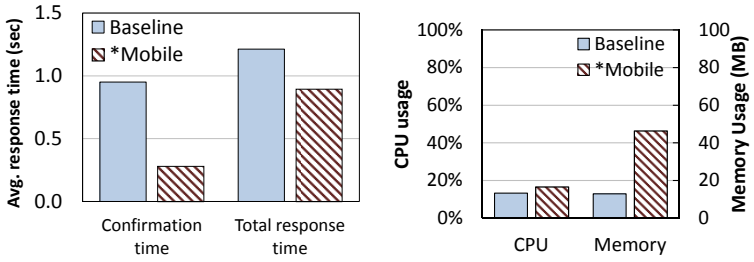
Figure 38: Comparison of PC version, mobile version, and **Mobile* of Georgia Tech T-Square

site hosted in the full-blown T-Square application. Figure 38(b) shows the same tool in the mobile version of T-Square. The mobile site still inherits majority of UI elements and layout from the PC version, thus is not optimized for the smartphones. Figure 38(c) shows the application view of the same tool in T-Square mobilized by **Mobile*. In both case studies, translation transformation service of *MORPH* allows **Mobile* to provide a more smartphone-friendly view than the mobile version with a considerably less development effort.

6.4.6 Interaction Response Time

An important requirement for usability of a UI of an application is feedback to the user for any interaction with the user interface. An example is when a user clicks on a button, an animation of the button is presented to the user indicating that the action of clicking the button is completed. While the actual application functionality performed by clicking the button might take some time depending on processing requirements, the animation of the button click gives an instant feedback to the user. We term the time taken to provide this feedback as the event confirmation

time. The time taken to complete the application functionality is known as the total interaction response time. The lower the confirmation time the better is the usability of an application. When using a thin client solution, feedback should come from the remote machine. Network latency can lead to an increased confirmation time. This is evidenced from Figure 39(a), which shows the average for the two time metrics over multiple actions of a task. Since **Mobile* uses local rendering of UI, the confirmation feedback can be shown to the user immediately. The total interaction time however is comparable for the two approaches because eventually the application functionality itself is performed by the remote machine.



(a) Interaction response time (b) Overhead analysis

Figure 39: Microscopic analysis of **Mobile*

6.4.7 Overhead Analysis

An important goal of our solution is to use minimal resources of the smartphone. We have already seen how our solution relies on a low network traffic footprint. We now characterize the CPU and memory resources used by our app on the Android smartphone (Figure 39(b)). From the figure we observe that **Mobile* uses comparable CPU resources with respect to the baseline solution. While the memory usage of **Mobile* is comparatively higher than the baseline solution, we argue that memory usage is still acceptable since recent smartphones typically are equipped with at least 512MB RAM.

CHAPTER VII

ADD-ON TRANSFORMATION SERVICES AND SYSTEM INTEGRATION

7.1 Overview

In the previous chapters, we have presented a rapid application delivery protocol called *super-aggregation* that effective leverages heterogeneous interfaces. We have also presented a remote computing for heterogeneous devices called *MORPH*, including view virtualization and three core transformation services. In this chapter, we present the remaining work that completes this dissertation. First, we describe the design of other add-on transformation services that can operate on the *virtual view* of *MORPH*. Second, we present and evaluate the integrated operation of *MORPH* and *super-aggregation*.

7.2 Add-on transformation services

In this section, we explain how the add-on transformation services introduced in Chapter 4 can be implemented with *virtual view*. All services discussed are implemented entirely at the backend and thus can be enabled dynamically without the need to make any changes to the frontend that has been installed on the smartphones.

7.2.1 Reduction

The reduction service allows the user to remove certain elements from a *virtual view* to realize a simpler user interface in the mobile app transformed by *MORPH*. The user can either manually choosing the useful UI elements to keep in the *virtual view* or let the reduction service automatically suggesting the frequently used subset of UI elements. For manual reduction, the service can provide an "edit mode" by adding

a "show/hide" toggle button adjacent to each UI element. For automated reduction, the service listens to the *OnActivity()* events to analyze user activity. In run-time, the service registers to the *OnOpen()* and *OnUpdate()* events to change the *status* attribute of the UI elements in a *virtual view* as shown or hidden.

7.2.2 Overflow

The overflow service improves the usability of a dense *virtual view* of lots of elements by intelligently splitting a *virtual view* into multiple views, including the main view and the overflow view. Only a set of number of elements are presented in the main view to suit the small screen of the smartphone, and the remaining elements are moved into "overflow" views that can be made visible on demand. Similar to the reduction service, the elements in the main view can be manually configured by the user, or automatically configured by the overflow service based on the usage frequency of the elements. There are two ways that an overflow view can be created for the remaining elements. The first one is a pop-up dialog that will be layered on top of the main view when invoked. The second one is a in-line expansion where the elements are stored in an invisible container, which is made visible when invoked. During run-time, the service registers to the *OnOpen()* and *OnUpdate()* events to reorganize the elements in the *virtual view*.

7.2.3 Zoom

The zoom service allows a user to adjust richness of the frontend with several zoom levels. The lowest zoom level provides the simplest user interface with the most limited set of features, and the highest zoom level enables all features that the user wants to access from a smartphone. Each element in a *virtual view* is associated to a zoom level. The zoom service registers to *OnOpen()* and *OnUpdate()* events and add a knob to allow the user to adjust the zoom level from the frontend. When the user switches to a certain zoom level, all elements at a higher zoom level is removed from

the *virtual view*, and then the updated *virtual view* is sent to the frontend.

7.2.4 Rearrangement

The rearrangement service allows the user to customize the order of UI elements presented in a *virtual view*. The service is realized by moving the nodes in the tree structure of the *virtual view* by writing the children attribute. Similar to the reduction service, rearrangement configuration can be done either manually or automatically, and either approach is supported by the same functions in the *virtual view* API. In run-time, the rearrangement service changes the order of elements in their parent based on the configuration.

7.2.5 Customized Translation

The customized translation service allows users to customize the look-and-feel of the mobile app transformed by *MORPH*. The service associates a look-and-feel template to each type of element in a virtual view. In an Android environment, the template can be defined as an XML snippet that describes the style settings such as size, foreground color, background color, padding, margin, layout alignment, etc. The customized translation service registers to *OnOpen()* and *OnUpdate()* events and provides the templates along with the *virtual view*.

7.3 Integrated operation of MORPH *and* super-aggregation

In this section, we present an experimental study on the integrated operation of the two proposed protocols, i.e. *MORPH* and *super-aggregation*. Since *super-aggregation* is designed as a generic layer-3.5 middleware solution, it enhances application delivery throughput independent from the application protocol on top of it. Thus, the network performance of *MORPH* can be enhanced by *super-aggregation* without any explicit modification.

We conduct integrated evaluation using trace-based experiments in a real testbed

shown in Figure 40. The client device is a Samsung Galaxy S smartphone running Google Android 2.1 with a 1GHz CPU and 512MB RAM. The server is a Windows Server 2003 machine hosted in Amazon EC2 cloud. As described in Chapter 6, the *MORPH* is prototyped in both the Windows server and the Android client using VNC as the underlying remote computing protocol. WAN emulator between the client and the Internet emulates a mobile network with both Wi-Fi and 3G connectivity. The emulator takes two traces collected from experiments conducted in a testbed with real Wi-Fi and 3G deployment, which is described in Chapter 3. The traces contain the instantaneous throughput of default TCP and *super-aggregation* (integrated operation of three design principles) in the same mobile network scenario used in Chapter 3. The bandwidth constraint of the WAN emulator is changed every one second based on the traces. The WAN emulator is the bottleneck of the end-to-end path since the smartphone connects to a nearby Wi-Fi at 54Mbps with excellent signal strength. Thus, the trace-based WAN emulation allows us to measure the network performance of a real smartphone application in a mobile network.

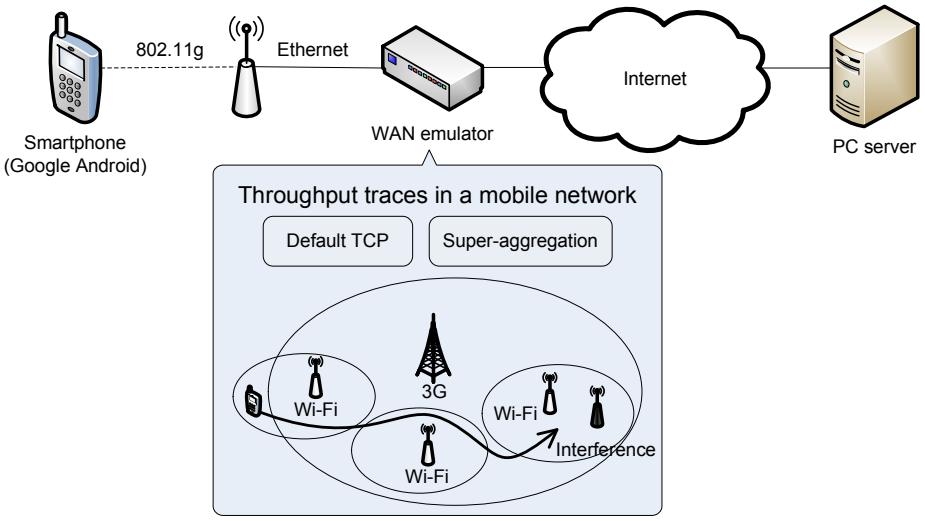


Figure 40: Testbed for integrated evaluation of *MORPH* and *super-aggregation*

Using the testbed, we evaluate the performance of a mobile application from a smartphone connecting to the Internet via a mobile network. The performance metric

is average response time of all actions performed in the mobile application. We consider two applications. One is accounting software called QuickBooks, which is a native PC application. The other is collaboration software called T-Square, which is a web application. We evaluate the performance of both *MORPH* and *super-aggregation* by comparing them to the respective baseline solutions. For *MORPH*, the baseline solution is VNC for the native application or Android mobile browser for the web application. For *super-aggregation*, the baseline is the default TCP mechanism of a smartphone, which switches from cellular data to Wi-Fi if an AP is accessible.

Figure 41 shows the average response time of accessing QuickBooks and T-Square from the smartphone via a mobile network using the four combinations of technologies. Baseline mobile app with default TCP has the highest response time. *Super-aggregation* reduces the average response time by almost half for both applications. The reduction comes from the improvement in throughput and quick recovery after disconnection with *super-aggregation*. *MORPH* provides improvement in response time by 51% and 68% for T-Square and QuickBooks, respectively. The improvement comes from the reduction in traffic by the traffic suppression service. Using *MORPH* and *super-aggregation* in tandem combines the benefits from both solution and achieves reduction in average response time up to 85%. The integrated evaluation shows that *MORPH* and *super-aggregation*, both individually and jointly, can significantly reduce the response time in accessing mobile applications from smartphones.

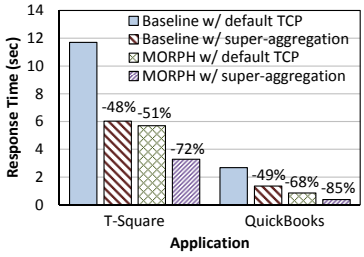


Figure 41: Response time in remote access to applications from a smartphone

CHAPTER VIII

CONCLUSION AND FUTURE WORK

In this dissertation, we focused on two important problems that arise in the emerging mobile computing platform of smartphones - application mobilization and application delivery. We identified the open challenges of both problems in the context of smartphone, which is distinct from the traditional PC platform in terms of network connectivity, user interface, resource availability, and software platform. In application mobilization, traditional approaches require significant time and effort, and they typically suffer from functionality scale back. In application delivery, performance is impacted by the inferior characteristics of the underlying mobile and wireless connectivity. In the following we summarize the main contributions of this dissertation research.

8.1 Main Contributions

- We presented a solution strategy called *super-aggregation* for mobile devices with multiple interfaces that achieves more than the sum of the parts in terms of aggregate performance when the multiple interfaces are used simultaneously.

- We showed that *super-aggregation* can be realized purely as a layer-3.5, mobile-device-only solution and how an instantiation of *super-aggregation* can be used to achieve TCP acceleration in wireless data networks.

- We implemented *super-aggregation* on two mobile platforms - a laptop and an Android mobile phone and in the process use real-world 3G and Wi-Fi wireless data networks to demonstrate the efficacy of the proposed *super-aggregation* principles and overall solution.

- We presented *MORPH*, a remote computing protocol for heterogeneous devices. *MORPH* improves user performance in remote computing from a smartphone by transforming application views in the PC platform into smartphone-friendly views. *MORPH* abstracts application views independent of the underlying application framework into a uniform representation called virtual views, which allows transformation services to opreate on them and realize smartphone-friendly views.

- We presented the design of a core transformation service of *MORPH* called aggregation that reduces the time and effort required in accomplishing tasks from smartphones. The solution called $MORPH_{Aggregation}$ is prototyped on top of VNC as the underlying remote computing protocol. $\text{MORPH}_{\text{Aggregation}}$ introduces a key building block called smart macros that allow users to record and replay their repetitive tasks. Smart macros have both the generality of traditional raw macros but at the same time possess the robustness of application macros.

- We presented a solution called **Mobile* that achieves rapid application mobilization. The solution is built on top of MORPH and consists of two core transformation services: (i) translation service presents a smartphone friendly view by converting each element in the virtual view into a native element in the smartphone, and (ii) trac suppression service that it intelligently suppresses and reduces traffic required for the remote access leveraging knowledge of the virtual view.

- We presented add-on transformation services that can be programmed onto the virtual view of *MORPH*. The add-on transformation services include reduction, overflow, zoom, rearrangement, and customized translation.

- We presented the integrated operation of *super-aggregation* and *MORPH*. We

conducted an experimental study to evaluate the performance of *super-aggregation* and *MORPH*, both individually and jointly, in accessing a mobile application from a smartphone.

8.2 Future Work

While the dissertation has identified several challenges in application mobilization and application delivery in the context of smartphones, there are several open problems that have opened up as a result of this dissertation research.

- The *super-aggregation* of this dissertation research focuses on Wi-Fi 802.11g and 3G in the design and prototyping of the proposed solution. The smartphone platform keeps evolving with new communication technologies, such as 802.11n and 4G. Extending the *super-aggregation* solution to more wireless protocols, such as 802.11n, 4G, Bluetooth, satellite networks would provide more practical contributions to application delivery of new smartphones.

- In the *super-aggregation* work presented in this dissertation, we presented three design principles that leverage three heterogeneous characteristics between Wi-Fi and cellular data networks: self-contention, disconnectivity, and random packet loss. There are more dimensions that different wireless networks may exhibit different characteristics. For example, cellular data networks are typically monitored by the telecom carriers, which may adopt certain QoS mechanisms to ensure the service their customers pay for. On the other hand, Wi-Fi networks typically provide free access with no or minimum QoS control. *Super-aggregation* can be extended with more design principles derived from other heterogeneous characteristics between wireless networks.

- PC applications are built without the awareness of contextual information of

the user, since the context of application usage typically remain static. The contextual information includes but is not limited to location, speed, or the person nearby. When *MORPH* enables transformation of PC applications into mobile apps for a smartphone, contextual information available at the smartphone can potentially be leveraged to make the mobilized applications more mobile-friendly. For example, printing is a common feature in most PC applications. When a user wants to print a document in a application from a smartphone, he or she may want to print the document to the closest printer instead of the default one configured in the system. The challenge of the contextualization service is to allow for the collection of contextual information on the mobile-device, and furnish it appropriately as input to the PC application on-demand, *even when the original application is not equipped for contextualization.*

- Different software vendors typically develop PC applications and integration of functionality from different applications cannot be achieved through conventional approaches without direct collaboration of application vendors. Application Federation is another transformation service that can potentially be realized in *MORPH*. It will allow the user to federate two or more different applications and present them (or a subset of their functionalities) as a single *federated application* on the smartphone. The federated application view allows the user to perform activity across multiple PC applications and/or get simultaneous view into them in a dashboard-like smartphone app.

- Tablets are an emerging class of mobile devices with hybrid characteristics borrowed from smartphones and PCs. Most mobile apps for smartphones can also run in tablets she most popular tablets in market today such as the iOS-based Apple iPad and the numerous Android-based tablets use the same underlying OS as their smartphone counterparts. However, the user interface of mobile

apps typically needs to be customized or optimized for tablets for best usability and performance. Thus, the *MORPH* frontends have to be carefully designed to provide users a consistent look and feel compared to other mobile apps they would use on those platforms. Also, performance considerations including processing and memory overheads of the frontend will have to be studied.

CHAPTER IX

PUBLICATIONS

Journal Papers

1. **C.-L. Tsao** and R. Sivakumar, "A Super-Aggregation Strategy for Multi-homed Mobile Hosts with Heterogeneous Wireless Interfaces," submitted to IEEE/ACM Transactions on Mobile Computing, 2011.

2. S.Lakshmanan, **C.-L. Tsao**, and R. Sivakumar, "Symbiotic Coding for high density Wireless LANs," submitted to IEEE/ACM Transactions on Mobile Computing, 2011.

3. S.Lakshmanan, **C.-L. Tsao**, and R. Sivakumar, "Aegis: Physical Space Security for Wireless Networks With Smart Antennas," IEEE/ACM Transactions on Networking, vol. 18, no. 4, pp. 1105-1118, August 2010.

4. Y. Jeong, S. Kakumanu, **C.-L. Tsao**, and R. Sivakumar, "VoIP over Wi-Fi Networks: Performance Analysis and Acceleration Algorithms," Springer Mobile Networks and Applications Journal (MONET), vol. 14, no. 4, pp. 523-538, August 2009.

Conference Papers

1. **C.-L. Tsao**, S. Kakumanu, and R. Sivakumar, "SmartVNC: An Effective Remote Computing Solution for Smartphones," ACM International Conference on Mobile Computing and Networking (MOBICOM), Las Vegas, Nevada, USA, Sept, 2011.

2. **C.-L. Tsao** and R. Sivakumar, "On Effectively Exploiting Multiple Wireless Interfaces in Mobile Hosts," ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT), Rome, Italy, Dec. 2009.

3. S. Lakshmanan, **C.-L. Tsao**, and R. Sivakumar, "On Coding Concurrent Transmissions in Wireless Networks," Poster Presentation, ACM International Conference on Mobile Computing and Networking (MOBICOM), Beijing, China, Spet. 2009.

4. S. Lakshmanan, **C.-L. Tsao**, R. Sivakumar, and K. Sundaresan, "Securing Wireless Data Networks against Eavesdropping using Smart Antennas," International Conference on Distributed Computing Systems (ICDCS), Beijing, China, June 2008.

5. Y. Jeong, S. Kakumanu, **C.-L. Tsao**, and R. Sivakumar, "Improving VoIP Call Capacity over IEEE 802.11 Networks," IEEE International Conference on Broadband Communications, Networks, and Systems (BROADNETS), Raleigh, NC, USA, Sept. 2007.

REFERENCES

[1] "android-vnc-viewer." http://code.google.com/p/android-vnc-viewer/.

[2] "App Mobi." `http://www.appmobi.com/`.

[3] "Appcelerator." `http://www.appcelerator.com/`.

[4] "Apple's App Store Downloads Top 10 Billion." http://www.apple.com/pr/library/2011/01/22appstore.html.

[5] "AutoHotkey." http://www.autohotkey.com/.

[6] "Automator in Mac OS X." http://developer.apple.com/macosx/automator.html.

[7] "The comscore 2010 mobile year in review." http://www.comscore.com/Press_Events/Presentations_Whitepapers/2011/2010_Mobile_Year_in_Review.

[8] "FeedCircuit." `http://feedcircuit.garage.maemo.org/`.

[9] "HyperSQL." http://hsqldb.org/.

[10] "IDC: Q4 2010 Smartphone Shipments Increase 87.2%." http://www.idc.com/about/viewpressrelease.jsp?containerId=prUS22689111.

[11] "iMacros." http://www.iopus.com/imacros/.

[12] "iTeleport JadduVNC." http://www.iteleportmobile.com/iphone.

[13] "KDE Accessibility Project." http://accessibility.kde.org/.

[14] "KDE Accessibility Project." `http://accessibility.kde.org/`.

[15] "LogMeIn." https://secure.logmein.com/.

[16] "Mac OS X Accessibility Protocol." http://developer.apple.com/library/mac/documentation/Accessibility/Conceptual/AccessibilityMacOSX/OSXAXModel/OSXAXmodel.html.

[17] "Mac OS X Accessibility Protocol." `http://developer.apple.com/library/mac/#documentation/Accessibility/Conceptual/AccessibilityMacOSX/OSXAXModel/OSXAXmodel.html`.

[18] "Microsoft Office." http://office.microsoft.com/.

[19] "Mochasoft VNC/RDP Clients for iPhone and Android." http://www.mochasoft.dk/.

[20] "PC-over-IP." http://www.teradici.com/.

[21] "Remote Desktop Protocol." http://msdn.microsoft.com/en-us/library/aa383015(VS.85).aspx.

[22] "Smartphones have conquered PCs." http://money.cnn.com/2011/02/09/technology/smartphones_eclipse_pcs/index.htm.

[23] "Smartphones to Overtake Feature Phones in U.S. by 2011." http://blog.nielsen.com/nielsenwire/consumer/smartphones-to-overtake-feature-phones-in-u-s-by-2011/.

[24] "Tasker for Android." http://tasker.dinglisch.net/.

[25] "TeamViewer." http://www.teamviewer.com/download/mobile.aspx.

[26] "The RFB Protocol." `http://www.realvnc.com/docs/rfbproto.pdf`.

[27] "TightVNC Software." `http://www.tightvnc.com/`.

[28] "UI Automation." `http://msdn.microsoft.com/en-us/library/ms747327.aspx`.

[29] "RFC 1122: Requirements for Internet Hosts - Communication Layers," 1989.

[30] Abd El Al, a., "LS-SCTP: a bandwidth aggregation technique for stream control transmission protocol," *Computer Communications*, vol. 27, pp. 1012–1024, June 2004.

[31] Agarwal, Y., Chandra, R., Wolman, A., Bahl, P., Chin, K., and Gupta, R., "Wireless wakeups revisited: energy management for voip over wi-fi smartphones," in *Proceedings of ACM MobiSys*, (New York, NY, USA), pp. 179–191, ACM, 2007.

[32] Allman, M., "RFC 3465: TCP Congestion Control with Appropriate Byte Counting (ABC)," 2003.

[33] Altman, E., Bp, I., and Ingeniera, F. D., "Novel delayed ack techniques for improving tcp performance in multihop wireless networks," in *Proceedings of Personal Wireless Communications*, pp. 23–25, 2003.

[34] AndroLib, "Android Market Statistics." http://www.androlib.com/appstats.aspx, 2011.

[35] Balakrishnan, H. and Katz, R., "Explicit loss notification and wireless web performance," in *Proceedings of the IEEE Globecom Internet Mini-Conference*, pp. 1–5, 1998.

[36] Balakrishnan, H., Seshan, S., Amir, E., and Katz, R. H., "Improving TCP/IP performance over wireless networks," in *Proceedings of ACM MobiCom*, (New York, NY, USA), pp. 2–11, ACM, 1995.

[37] Baratto, R. A., Potter, S., Su, G., and Nieh, J., "MobiDesk : Mobile Virtual Desktop Computing Categories and Subject Descriptors," in *MobiCom*, 2004.

[38] Beverly, R. and Bauer, S., "The spoofer project: Inferring the extent of source address filtering on the Internet," in *Proceedings of the Steps to Reducing Unwanted Traffic on the Internet*, 2005.

[39] Bila, N., Ronda, T., Mohomed, I., Truong, K. N., and Lara, E. D., "PageTailor : Reusable End-User Customization for the Mobile Web," in *MobiSys*, 2007.

[40] Bolin, M., Webber, M., Rha, P., Wilson, T., and Miller, R. C., "Automation and customization of rendered web pages," in *UIST*, 2005.

[41] Bychkovsky, V., Hull, B., Miu, A., Balakrishnan, H., and Madden, S., "A measurement study of vehicular internet access using in situ wi-fi networks," in *Proceedings of ACM MobiCom*, (New York, NY, USA), pp. 50–61, ACM, 2006.

[42] Chang, T.-Y., Velayutham, A., and Sivakumar, R., "Mimic: raw activity shipping for file synchronization in mobile file systems," in *MobiSys*, 2004.

[43] Cypher, A., Halbert, D. C., Kurlander, D., Lieberman, H., Maulsby, D., Myers, B. A., and Turransky, A., eds., *Watch what I do: programming by demonstration*. Cambridge, MA, USA: MIT Press, 1993.

[44] de Cuetos, P. and Ross, K. W., "Adaptive rate control for streaming stored fine-grained scalable video," in *Proceedings of NOSSDAV*, (New York, NY, USA), pp. 3–12, ACM, 2002.

[45] Dunigan, T. and Fowler, F., "A tcp-over-udp test harness," Tech. Rep. ORNL/TM-2002/76, Oak Ridge, TN, 2002.

[46] Fracchia, R., Casetti, C., Chiasserini, C.-F., and Meo, M., "WiSE: Best-Path Selection in Wireless Multihoming Environments," *IEEE Transactions on Mobile Computing*, vol. 6, pp. 1130–1141, Oct. 2007.

[47] Goff, T., Moronski, J., Phatak, D. S., and Gupta, V., "Freeze-TCP: a true end-to-end TCP enhancement mechanism for mobile environments," in *Proceedings of IEEE INFOCOM*, vol. 3, (Tel Aviv), pp. 1537–1545, Mar. 2000.

[48] Henderson, T. and Katz, R., "Transport protocols for Internet-compatible satellite networks," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 2, pp. 326–344, 1999.

[49] Holleis, P., Otto, F., Hussmann, H., and Schmidt, A., "Keystroke-level model for advanced mobile phone interaction," in *CHI*, 2007.

[50] Hsieh, H.-Y., Kim, K.-H., Zhu, Y., and Sivakumar, R., "A receiver-centric transport protocol for mobile hosts with heterogeneous wireless interfaces," *Proceedings of the 9th annual international conference on Mobile computing and networking - MobiCom '03*, p. 1, 2003.

[51] Hsieh, H.-Y. and Sivakumar, R., "A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts," *Wireless Networks*, vol. 11, no. 1-2, pp. 99–114, 2005.

[52] Hupp, D. and Miller, R. C., "Smart Bookmarks : Automatic Retroactive Macro Recording on the Web," in *UIST*, 2007.

[53] Iyengar, J., Amer, P., and Stewart, R., "Concurrent Multipath Transfer Using SCTP Multihoming Over Independent End-to-End Paths," *IEEE/ACM Transactions on Networking*, vol. 14, pp. 951–964, Oct. 2006.

[54] Kim, K.-H. and Shin, K. G., "PRISM: Improving the performance of inverse-multiplexed TCP in wireless networks," *IEEE Transactions on Mobile Computing*, vol. 6, pp. 1297–1312, Dec. 2007.

[55] Kurlander, D. and Feiner, S., "A history-based macro by example system," in *UIST*, (New York, New York, USA), pp. 99–106, 1992.

[56] Kushman, N., Brodsky, M., Dina, S. R. K. B., Regina, K., and Rinard, M., "WikiDo," in *HotNets*, 2009.

[57] Laing, A., Craig, D., and White, A., "Vision Statement: High-Performance Office Space," *Harvard Business Review*, Sept. 2011.

[58] Lamberti, F. and Sanna, A., "Extensible GUIs for remote application control on mobile devices," *IEEE computer graphics and applications*, vol. 28, no. 4, pp. 50–7, 2008.

[59] Leshed, G., Haber, E. M., Matthews, T., Lau, T., Ave, C., Rd, H., and Jose, S., "CoScripter : Automating & Sharing How-To Knowledge in the Enterprise," in *CHI*, 2008.

[60] Lieberman, H., ed., *Your wish is my command: programming by example*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[61] Likert, R., "A technique for the measurement of attitudes.," *Archives of Psychology*, vol. 22, no. 140, pp. 1–55, 1932.

[62] Magalhaes, L. and Kravets, R., "Transport level mechanisms for bandwidth aggregation on mobile hosts," in *Proceedings of INCP*, pp. 165–171, Nov. 2001.

[63] Mickens, J., Elson, J., and Howell, J., "Mugshot : Deterministic Capture and Replay for JavaScript Applications," in *NSDI*, 2010.

[64] Mohomed, I., "Enabling mobile application mashups with Merlion," in *HotMobile*, 2010.

[65] Moshchuk, A., Gribble, S. D., and Levy, H. M., "Flashproxy : Transparently Enabling Rich Web Content via Remote Execution," in *MobiSys*, 2008.

[66] Nichols, J., Hua, Z., and Barton, J., "Highlight : A System for Creating and Deploying Mobile Web Applications," in *UIST*, 2008.

[67] Pering, T., Agarwal, Y., Gupta, R., and Want, R., "Coolspots: reducing the power consumption of wireless mobile devices with multiple radio interfaces," in *Proceedings of ACM MobiSys*, (New York, NY, USA), pp. 220–232, ACM, 2006.

[68] Phatak, D. and Goff, T., "A novel mechanism for data streaming across multiple IP links for improving throughput and reliability in mobile environments," in *Infocom*, Ieee, 2002.

[69] Radha, H. M., van der Schaar, M., and Chen, Y., "The MPEG-4 fine-grained scalable video coding method for multimedia streaming over IP," *IEEE Transactions on Multimedia*, vol. 3, pp. 53–68, Mar. 2001.

[70] Rahmati, A. and Zhong, L., "Context-for-wireless: context-sensitive energy-efficient wireless data transfer," in *Proceedings of ACM MobiSys*, (New York, NY, USA), pp. 165–178, ACM, 2007.

[71] RealVNC Ltd, "The RFB Protocol." http://www.realvnc.com/docs/rfbproto.pdf.

[72] Rodriguez, P., Chakravorty, R., Chesterfield, J., Pratt, I., and Banerjee, S., "Mar: a commuter router infrastructure for the mobile internet," in *Proceedings of ACM MobiSys*, (New York, NY, USA), pp. 217–230, ACM, 2004.

[73] Sharma, P., Lee, S.-J., Brassil, J., and Shin, K. G., "Aggregating bandwidth for multihomed mobile collaborative communities," *IEEE Transactions on Mobile Computing*, vol. 6, pp. 280–296, Mar. 2007.

[74] Sinha, P., Nandagopal, T., Venkitaraman, N., Sivakumar, R., and Bharghavan, V., "WTCP: a reliable transport protocol for wireless wide-area networks," *Wireless Networks*, vol. 8, no. 2/3, pp. 301–316, 2002.

[75] Snoeren, A. C., "Adaptive inverse multiplexing for wide-area wireless networks," in *Global Telecommunications Conference, 1999. GLOBECOM '99*, vol. 3, (Rio de Janeireo), pp. 1665–1672, 1999.

[76] Sugiura, A. and Koseki, Y., "Simplifying macro definition in programming by demonstration," in *UIST*, (New York, New York, USA), 1996.

[77] Zhang, D., "Web content adaptation for mobile handheld devices," *Communications of the ACM*, vol. 50, pp. 75–79, Feb. 2007.